Best
Available
Copy

A D [illegible]

BN (1)

AD-A155 458

Progress Report

ARTIFICIAL INTELLIGENCE — RESEARCH AND APPLICATIONS

By: MEMBERS OF THE PROJECT STAFF

Edited by: NILS J. NILSSON

Prepared for:

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY
ARLINGTON, VIRGINIA 22209

CONTRACT DAHC04-75-C-0005

DTIC FILE COPY

Approved for public release; distribution unlimited.

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION IS UNLIMITED (A)

DTIC
SELECTE
JUN 1 7 1985
S D
G

STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

85 06 13 011

STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

Approved for public release;
distribution unlimited.

Form Approved
Budget Bureau No. 22-R0293

May 1975

Progress Report
Covering the Period 9 March 1974 through 31 March 1975
Stanford Research Institute Project 3805

ARTIFICIAL INTELLIGENCE — RESEARCH AND APPLICATIONS

Accession For
NTIS GRA&I ☒
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A

Edited by

NILS J. NILSSON
Project Leader
(415) 326-6200, Ext. 2311

With Contributions by Members of the Project Staff

CONTRACT DAHC04-75-C-0005
ARPA Order Number 2894
Program Code Number 61101E

Effective Date of Contract: 10 October 1974
Contract Expiration Date: 9 October 1975
Amount of Contract: $750,837

DTIC COPY INSPECTED

Prepared for

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY
ARLINGTON, VIRGINIA 22209

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

Approved by:

BERTRAM RAPHAEL, *Director*
Artificial Intelligence Center

BONNAR COX, *Executive Director*
Information Science and Engineering Division

Copy No. ....

MEMBERS OF THE PROJECT STAFF

N. Nilsson, Project Leader

G. Agin

H. Barrow

A. Brain

B. Deutsch

R. Duda

R. Fikes

T. Garvey

P. Hart

G. Hendrix

D. Lynch

B. Meyer

M. Rattner

E. Sacerdoti

D. Sagalowicz

G. Sutherland

J. Tenenbaum

B. Wilber

J. Zegers

ABSTRACT

This report describes the work performed during the last year on a continuing project to develop a computer based consultant (CBC) system. The system is being designed to talk (in ordinary English) with a human user to help him perform tasks entailing maintenance and troubleshooting of electromechanical equipment.

Our plan for this research calls for building a series of increasingly sophisticated systems to be demonstrated in 1976, 1977, and 1978. Our current (April 1975) demonstration system is a base for future systems and already illustrates abilities to interact with an apprentice to help him assemble a small air compressor.

The body of the report is divided into two major sections. One of these describes the technology behind the April 1975 demonstration system. The other describes work in progress that will contribute to the power and versatility of future demonstration systems.

Our 1975 system has the following specific abilities: It can generate and execute plans for assembly/disassembly at several levels of detail. It can answer queries from the apprentice about the status of the equipment. It can point at parts of the compressor and can name parts pointed to by the apprentice. It has a rudimentary ability for two-way communication using speech. The basis for each of these is described in detail.

work supporting subsequent systems has been in the area of natural language understanding, modeling, troubleshooting, and vision. Separate subsections of the report deal with each of these areas.

The report also describes progress on some supporting activities including the SRI Artificial Intelligence Center Computer Facility, the language QLISP, hardware interface work, and a scanning laser range finder. It concludes with a list of presentations and publications by the project staff.

CONTENTS

ILLUSTRATIONS

TABLES

I. INTRODUCTION

A. The Computer-Based Consultant Project

1. Project Goal

For the past 18 months or so we have been working on a system that we call the Computer-Based Consultant (CBC). The system will be able to talk (in natural language) with a human user to help him perform tasks in some particular task environment. We intend to build a consultant that approximates the communication, perceptual reasoning, and factual knowledge skills of an actual expert on the scene.

Our main goal is to create the fundamental technology needed to build such consultant systems. We expect (or hope) that a good portion of this technology will be independent of the details of the particular kind of expertise being dealt with and of the details of the particular task environment. Thus we view our work as having potentially high payoff because of the great variety of applications in several task environments in which consulting expertise is needed or would be useful.

2. Background

The technology of computer-based consultants can be viewed as stemming from the confluence of two lines of research. One line has centered on formulating and encoding a great deal of knowledge about a chosen problem domain in order to produce a program whose performance rivals expert humans. Often cited examples of this research include programs that analyze chemical structure [1,2],* perform symbolic integration [3], or play board games [4,5,6].

The second line of research has focussed on methods for constructing a program that can carry on a dialog with a user. Important contributions to this research have come from work in computer aided instruction, and from work in understanding typed and spoken natural language. Representative examples of this work include programs to carry out a "mixed initiative" tutorial dialog [7,8]; to engage in a dialog about a toy block world [9]; and to understand spoken English sentences about such diverse topics as plumbing [10], news stories [11], moon rocks [12], or submarines [13].

Perhaps one of the best examples to date of a small computer-based consultant is the MYCIN system [14]. This system provides advice to physicians on the diagnosis and therapy of certain classes of bacterial diseases. It solicits various kinds of medical data from a physician user, can answer his questions (expressed in a highly restricted natural English format), and can accept advice from him regarding generally useful rules for diagnosis and therapy.

----

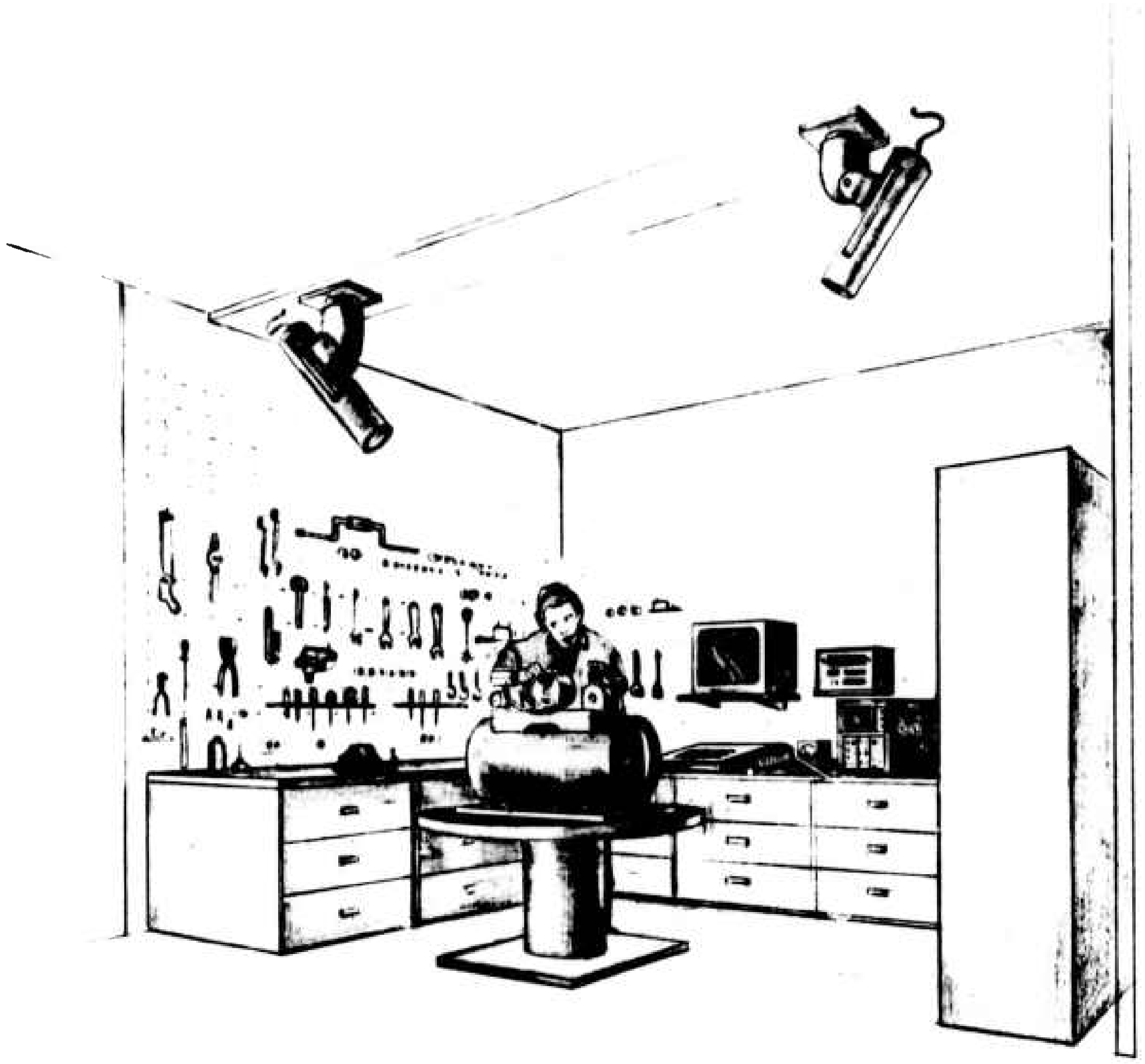
*References are listed at the end of this report.

3. The Task Environment

Notwithstanding our goal of developing fundamental and widely useful technology, we must, of course, select some particular task environment for the CBC system. We have attempted to select a task environment that is an important application area in its own right as well as one that serves well as a typical representative of a wide variety of applications.

We have selected the task of repairing, modifying, and checking out complex electromechanical equipment. Our CBC system will be helping an inexperienced mechanic (whom we shall call an apprentice) as he works on a piece of equipment in a "workstation." The kinds of problems encountered using this particular task environment are typical of those of many other environments. Furthermore, maintenance of equipment is an important task in itself, costing literally billions of dollars each year. Thus, there is a high potential for substantial savings deriving from improved efficiencies in this one application area alone.

Before discussing the details and specifications of the CBC system itself, let us first briefly consider some of the characteristics of equipment maintenance as a task area. Imagine an apprentice working on a piece of equipment in a workstation like the one sketched in Figure 1. Typically he is responsible for a variety of jobs, such as troubleshooting, repairing, or modifying equipment.



SA-1530-41

FIGURE 1 A WORKSTATION

In order to do these jobs, he needs certain kinds of specialized knowledge; he must know about the use of various tools, about principles of troubleshooting, and about principles of assembly and disassembly, and he must also know a certain amount of detail about the construction and operation of the specific equipment on hand.

A traditional way of conveying this knowledge to a mechanic has been through the use of manuals. A more nearly ideal, though usually impractical, way would be to make an expert human mechanic continuously available as a consultant. This expert could identify various components, answer specific questions about equipment details, suggest troubleshooting sequences, hypothesize causes of failure, warn of hazards, and so forth.

In order to begin to explore what would be involved in replacing the human expert by a computer-based expert, we recorded a number of dialogs between expert human mechanics and novice mechanics. The dialogs concern the air compressor shown in Figure 2. Two excerpts from these dialogs are presented below. At the time the dialogs were recorded, the expert and novice were in different rooms and the expert viewed the scene only by means of still pictures taken through a television camera. (We did this to simulate the limited visual information likely to be available to a computer based expert.) The first excerpt concerns the subtask of installing a pump pulley on the pump.

Excerpt 1

.

.

.

Expert: The pump pulley should be next.

Apprentice: Yes ... uh, does the side of the pump pulley with the holes face away from the pump or towards it?

E: Away from the pump.

A: All right.

E: Did you insert the key--that is, the half-moon shaped piece?

A: Yes, I did.

E: Be sure you check the alignment of the two pulleys before you tighten the set-screws.

A: Yes, I'm just now fiddling with that.

E: OK.

A: Tightening the Allen screw now.

E: OK, thank you.

.

.

.

This fragment illustrates several important abilities of consultants that contrast sharply with a static information source like a manual. First, notice that a question from the apprentice is answered directly and in his terms. There is no need for him to search through a mass of information, or to convert information from an abstract or "standard" form into a directly usable form. Notice that the expert is checking on progress by offering warnings and reminders about critical steps. This has the function not only of minimizing errors, but also of allowing the expert to keep track of the progress of the work. The latter function is the basis for the expert's ability to present relevant advice, and to present it in a context that is familiar to the apprentice.

The second dialog excerpt concerns the same subtask, but was carried out with different participants. It offers an interesting comparison of the different demands imposed by different skill levels:

Excerpt 2

.

.

.

Expert: Install the pulley on the shaft.

Apprentice: What is the first thing to do in installing the pulley?

E: Rotate the shaft so that the slot (keyway) is on the top.

A: OK ... now what?

E: Place the key in the slot.

A: Flat side upward?

E: Yes.

.

.

.

This short fragment dramatically illustrates the ability of the expert to descend into detailed instructions in order to help a very naive user. This apprentice needs much more help than the first one did, a situation foreshadowed by his initial question about a relatively simple operation.

These short dialog excerpts exemplify some of the abilities that a consultant needs in order to be helpful to the apprentice mechanic. Both introspection and protocol experiments point out a number of other required abilities, among which are the ability to provide advice about troubleshooting; to describe the use of tools; to describe the appearance of tools (or to be able to point them out); and, of course, the ability to use language. (We shall have more to say in a later section about how we are using protocols of this sort to design the subsystem for processing natural language.)

During the past year we have been using the simple air compressor of Figure 2 as an exemplar piece of equipment. While the device appears to be reasonably simple, certain of its subassemblies are rather complex. For example, the air pump, shown in Figure 3, contains a large number of parts put together in complex fashion.

The compressor has served us well as a beginning device, but we are now ready to move on to some more complex devices. We are especially anxious to select a device demanding rather sophisticated troubleshooting skills. We expect to make a decision soon about which new device to use. Several candidates are discussed in the Appendix.

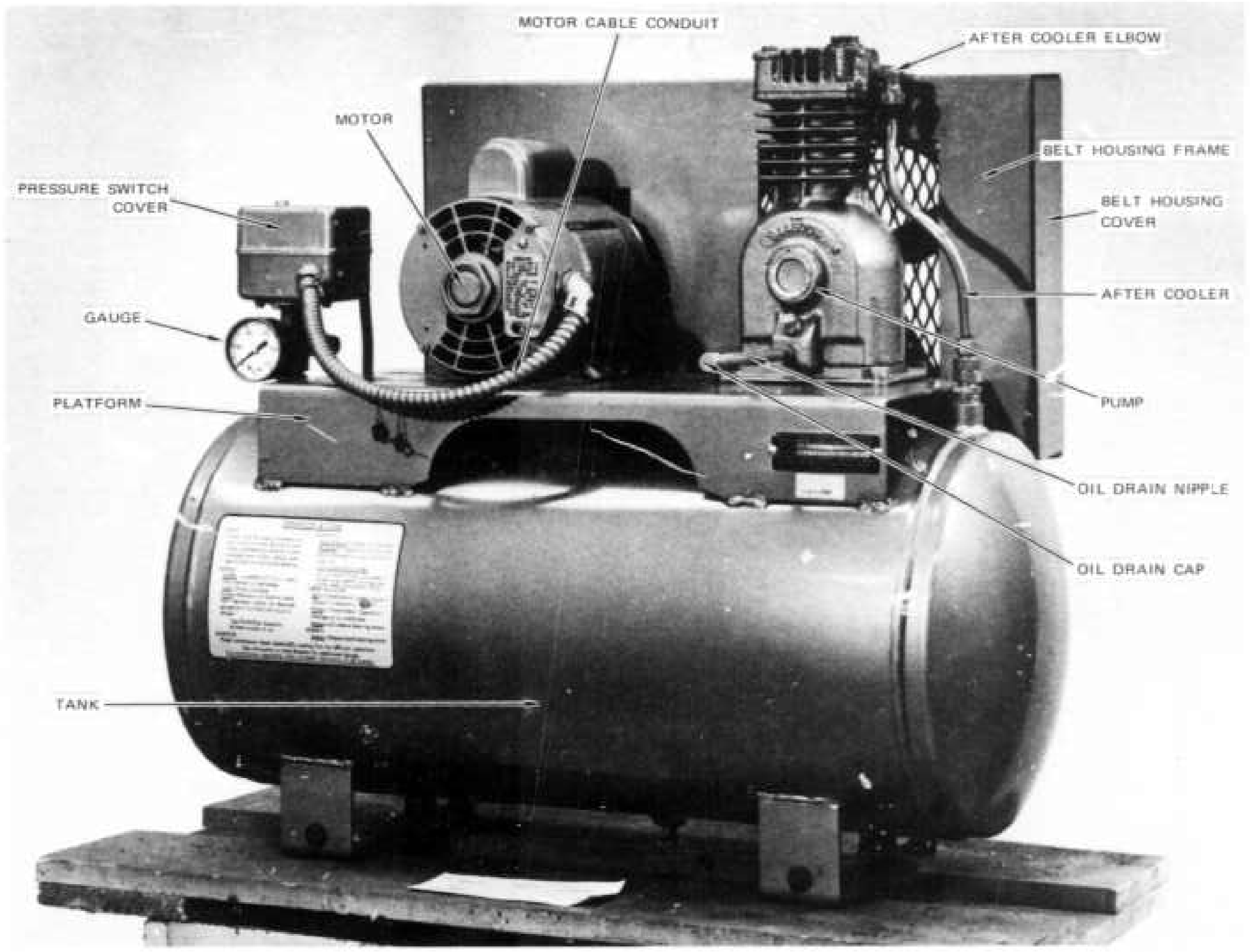
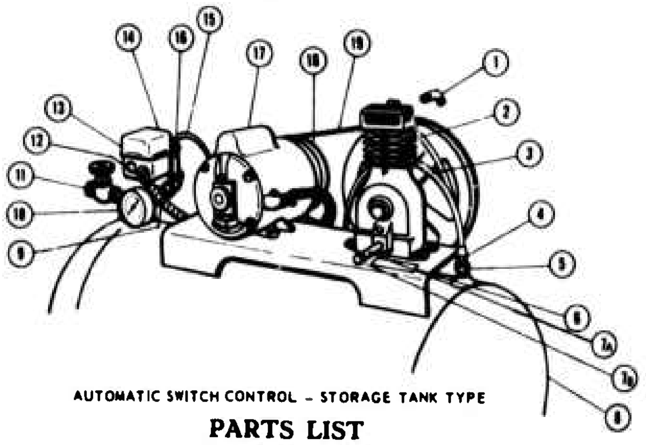


FIGURE 2 A SMALL AIR COMPRESSOR

Model Numbers

(17199) 102.17200	102.17220 (17218)	102.17021
(17209) 102.17210	102.17001	102.17046
	102.17011	102.17041



AUTOMATIC SWITCH CONTROL – STORAGE TANK TYPE

PARTS LIST

Reference Number	Part Number	NAME OF PART
1	690	Aftercooler Elbow ⅜" Flare Tube x ¼" Pipe
2	-	Pump – See Figure 9
3	37055	Aftercooler for 102.17001, 102.17011, 102.17021, 102.17046 and 102.17041
3	37057	Aftercooler for (17199) 102.17200, (17209) 102 17210, and 102.17220 (17218)
4	98556	Elbow Connector–¼" Tube x ⅜" Pipe Thread
5	15681	Check Valve–See Figure 4 for Parts
6	12776	Nameplate
7a	9609	Oil Drain Nipple–⅜" x 5" Long
7b	11292	Oil Drain Cap–⅜"
8	33779	Tank and Platform Assembly for (17199) 102.17200, (17209) 102 17210, 102.17220 (17218)
8	33780	Tank and Platform Assembly to Massachusetts Specifications for (17199) 102.17200, (17209) 102.17210, and 102.17220 (17218)
8	43781	Tank and Platform Assembly 102.17001 and 102.17011. Use for Massachusetts also.
8	43782	Tank and Platform Assembly for 102.17021, 102.17046 and 102.17041
8	43783	Tank and Platform Assembly to Massachusetts Specifications for 102.17021, 102.17046 and 102.17041
9	16614	Manifold for mounting Pressure Switch, etc. for 102.17021, 102.17046 and 102.17041. Use for all Mass. Specifications
9	33724	Manifold for mounting Pressure Switch and Gauge, (Safety Valve and Discharge Valve included) for (17199) 102.17200, (17209) 102.17210, 102.17220 (17218) and 102.17001. Not for Massachusetts Specificationn
10	12571	Pressure Gauge for 102.17001, 102.17011, 102.17021, 102.17046 and 102.17041
10	15793	Pressure Gauge for (17199) 102 17200, (17209) 102.17210, and 102 17220 (17218)
10	17088	Pressure Gauge for State of Massachusetts
11	95939	Discharge Valve for 102.17001, 102.17011, 102.17021, 102.17046 and 102.17041. Also used on Mass. Specification models (17199) 102.17200, (17209) 102 17210 and 102 17220 (17218)
12		BX conduit–Available locally (See Installation Instructions for wire size)
13	12191	Pressure Switch for (17209) 102.17210, 102 17220 (17218) 102.17011, 102.17021 and 102.17041
13	13858	Pressure Switch for 102.17046
13	32473	Pressure Switch for (17199) 102.17200
14	-	Pressure Switch Relief Valve. When ordering Relief Valve give Pressure Switch Nameplate Data. (Ref. No. 13)
15	18591	Check Valve Relief Tube (also available locally ¼" aluminum or copper tube 31½ long.
16	18550	Safety Valve for 102.17001, 102.17011, 102.17021, 102.17046 and 102.17041
16	18544	Massachusetts Safety Valve for (17199) 102 17200, (17209) 102.17210, 102.17220 (17218), 102 17001, 102.17011, 102.17021, 102.17046 and 102.17041
17	-	Motor or Gas Engine–See Motor Nameplate and Gas Engine Part Booklet for characteriarics; when writing regarding motor or engine service include complete nameplate data.
18	45053	Motor Pulley for (17199) 102.17200
18	45058	Motor Pulley for (17209) 102.17210 and 102.17021
18	45055	Motor Pulley for 102.17001 and 102.17011
18	45059	Motor Pulley for 102 17220 (17218) and 102 17041
18	30545	Motor Pulley for 102.17046
19	17469	Belt for 102.17001 (½" wide n 44" long)
19	17470	Belt for 102 17220 (17218), 102.17021 and 102 17041 (½" wide n 47" long)
19	18356	Belt for (17209) 102 17210 (½" wide n 46" long)
19	18481	Belt for 102.17011 and 102.17046 (½" wide n 45" long)
19	32622	Belt for (17199) 102 17200 – ½" wide n 43" long
Not Shown	96570	Aftercooler Vent Coco-Gas Engine Only
Not Shown	96800	Tank Drain Valve
Not Shown	13231	Tank Drain Valve Only for State of Mass.
Not Shown	43429	Cord and Plug Assembly for 102.17301, 102.17400 and 102.17410
Not Shown	43430	Adapter Ground Plug for 102.17304 102.17400 and 102.17410

SA-1530-56

FIGURE 3 PARTS LIST FOR AIR COMPRESSOR

4. Project Organization and Method of Approach

We have already stated that our goal for the CBC project is to build a system that approximates a human expert. It is rather difficult for us now to predict precisely just how fine or coarse this approximation will be at any given future time. In our early plans for the project [15] we stated some general specifications for a system we believed would be demonstrable in 1978. Current progress on the project reinforces our confidence in being able to achieve an impressive system by that time.

In planning our work we have relied heavily on studying tape-recorded dialogs between human experts and apprentices. From these we have extracted various requirements for our 1978 demonstration system. In summary these are:

(1) The system will be able to plan logical sequences of actions to accomplish goals. These goals will include a range of tasks involving check-out, maintenance, troubleshooting, repair, and assembly/disassembly of equipment.

(2) These sequences of actions will be planned to whatever level of detail is appropriate to the apprentice's ability and to the other requirements of the system.

(3) The system will be able to replan on encountering unexpected situations (such as the apprentice performing a task incorrectly or out of sequence).

(4) It will be able to answer specific questions about equipment, tasks, plans, and about its own knowledge.

(5) It will understand spoken utterances (with various restrictions appropriate to the state of the art of speech understanding).

(6) It will give instructions and answers to queries by speaking.

(7) It will be able to obtain information visually about the actions of the apprentice and the changing states of the equipment.

(8) It will point at things and create displays on request from the apprentice.

(9) It will be programmed in such a way that it will be easy to add more detailed knowledge about its specific domain of expertise or to add new knowledge about related domains.

In order to achieve these abilities, a substantial body of research must be undertaken. We have grouped the major research questions into four categories: problem-solving, natural language, vision, and system integration. Each category has its own team of researchers, although there is considerable overlap and sharing of personnel between each of these tasks. Our strategy is that as research progresses on the first three of these topics, the interim results will immediately be integrated into a growing demonstration system. Thus, at any time, the current version of the demonstration system can be taken as a measure of progress toward our 1978 goals.

The growing demonstration sytem will be an important tactic to ensure proper communication between various parts of the project. We are also using it to structure major subgoals and scheduling. We have defined certain subsidiary abilities that should be achieved by each of three major demonstrations to occur each April in 1975, 1976, and 1977 [16].

A description of the abilities of the CBC system at the time this report went to press is given in the following paragraphs. Subsequent sections of the report will describe the details of the technology underlying this present (April 1975) system and the work being done in support of future systems.

B. The 1975 Demonstration System

1. General

In this section we shall describe a demonstration that exhibits the current abilities of the system. We begin by describing the setting of the demonstration.

The place where the action takes place is called a workstation. It is a room, approximately 13 feet by 18 feet in size containing the following items:

A workbench with a tool box and tools.

A round table with a turntable top on which is placed a small air compressor; this is located in the middle of the room with access from all sides.

A computer terminal.

A microphone headset with a long cord that will reach to any point in the room.

A speaker/amplifier.

A TV camera, mounted near the ceiling on a movable pan/tilt head.

A TV display (RAMTEK) on which is displayed a TV picture of the air compressor, with a superimposed line drawing of the air compressor.

A laser/rangefinder mounted near the TV camera.

The air compressor is in a partially disassembled condition. The belt housing cover and belt are removed and lying on the workbench. The pump bolts are removed, and the pump is turned away from its normal orientation.

The apprentice enters the room and walks to the terminal and finds the following message: "Your goal is 'Assembled Aircompressor'. Please put on the headset and say 'Ready' into the microphone. The computer will give you verbal instructions. You can say any of the following responses:

OK indicates you can do the task.

HOW indicates you need more detailed instructions.

WHY indicates you want to know the motivation for the particular instruction just received.

HUH or WHAT or PLEASE-REPEAT will cause the last command to be repeated.

WHERE IS THE ... or SHOW ME THE ... followed by the name of a component will result in positioning of the laser rangefinder beam on that component.

WHAT IS THIS, coupled with your touching a component with the lighted wand, will get you the name of the component.

BREAK, PAUSE, or WAIT will cause an interrupt in the program execution so that you may use the terminal to query the program about the state of things. (Sorry, there is no voice I/O for these queries).

LOCATION OF ... followed by the name of a component will result in positioning of the laser beam on the location where the component is to be positioned.

CALIBRATE invokes a calibration sequence for camera, laser, and ranger.

FIND COMPRESSOR causes the laser/ranger to determine the location and orientation of the compressor and update its models and displays accordingly.

HELP means that something has gone wrong. We will examine the preceding steps and get you an expert person if necessary.

Please adjust the headset and proceed when you are ready."

2. Qualitative Abilities of the Computer Based Consultant

The abilities of the CBC are alluded to in the list of responses given above, and more details will be presented in Section II. In summary, the CBC contains a system for planning assembly or disassembly of the air compressor. The plan is represented by a structure called a procedural net which, in this demonstration, is built from a prespecified disassembled state of the compressor, namely the one described above. Each step of the assembly plan can be given at several different levels of detail, depending on the needs of the particular apprentice. The program keeps an internal model of the "connectedness" of the different components of the air compressor. It also contains a graphical (absolute positions) model of the locations of the different components. At the present time these models are limited. The graphical model "knows about" the pump, motor, pressure switch, belt housing frame, belt housing cover, pump pulley, motor pulley, platform, tank, and table. The connectivity model "knows about" all the visible components, but the two models do not as yet interact with each other.

The CBC is able to give responses using the voice of the Votrax Phoneme Synthesizer (described in detail in Section II.F.2 below) and can understand spoken instructions using the VIP-100 phrase recognizer (described in Section II.F.1). The TV camera is used in obtaining the location of the brightest point in its field of view. This is useful in cooperation with the laser beam, both in calibrating the graphical system and also in answering questions about names of particular parts of the compressor. The laser beam is program controlled, and can be moved to point at a given component, or to indicate calibration points.

When these various abilities are combined, they produce a system that is already rather unique and impressive. Further development is needed for each ability, and the system-building task is an ongoing one (see Section II.F). Other abilities will be added as they become available. In particular these are a natural language capability (see Section III.B) and a use of vision (scene analysis) to aid the problem solving (see Section III.D).

3. Example of a CBC/Apprentice Interaction

The following simulated dialog between an expert (E) and an apprentice (A) shows the abilities of the system as of April 1975. The interaction is by voice, with use of commercially available devices for voice recognition and voice synthesis.

E: TRY TO ACHIEVE ASSEMBLED AIRCOMPRESSOR

A: HOW?

E: THE TOP LEVEL PLAN IS INSTALL PUMP, INSTALL PUMPBRACE, INSTALL PUMPPULLEY, INSTALL BELT, AND INSTALL BELTHOUSINGCOVER

E: READY FOR QUESTIONS

A: SHOW ME THE PUMP

(Laser beam moves to point at pump)

E: READY

A: SHOW ME THE MOTOR

(Laser beam moves to point at motor)

E: READY

A: WHERE IS THE BELTHOUSINGFRAME?

(Laser beam moves to point at the belthousingframe)

E: READY

A: WHAT IS THIS? (Touching tank platform with wand)

E: TANK PLATFORM

E: READY

A: WHAT IS THIS? (Pointing with wand at pressure switch)

E: PRESSURE SWITCH COVER

E: READY

A: OK

E: INSTALL PUMP

A: HOW?

E: CONNECT PUMP TO PLATFORM

A: HOW?

E: POSITION PUMP ON PLATFORM

A: LOCATION OF PUMP?

(Laser beam moves to point at pumpplace on platform)

A: OK

E: POSITION PUMP ON PLATFORM

A: HOW?

E: PLACE THE PUMP ON THE PLATFORM IN SUCH A WAY THAT HOLES
IN THE PUMP ARE MATCHED UP WITH HOLES IN THE PLATFORM

A: OK

E: ATTACH PUMP TO PLATFORM

A: HOW?

E: USE BOLTS TO ATTACH PUMP TO PLATFORM

A: HOW?

E: GET 4 ONE AND A QUARTER INCH # 5/16 HEXAGONAL BOLTS AND
4 5/16 HOLE DIAMETER FLAT WASHERS AND 4 SIZE 9/16 HEXAGONAL NUTS

A: HUH?

(Apprentice did not hear the command)

E: GET 4 ONE AND A QUARTER INCH # 5/16 HEXAGONAL BOLTS AND
4 5/16 HOLE DIAMETER FLAT WASHERS AND 4 SIZE 9/16 HEXAGONAL NUTS

A: HOW?

E: GET 1 ONE AND A QUARTER INCH # 5/16 HEXAGONAL BOLT

A: ALL RIGHT

E: PLEASE REPHRASE YOUR ANSWER

(Expert did not recognize the response)

A: OK

E: GET ONE 5/16 HOLE DIAMETER FLAT WASHER

A: OK

E: GET ONE SIZE 9/16 HEXAGONAL NUT

A: OK

E: GET 3 ONE AND A QUARTER INCH # 5/16 HEXAGONAL BOLTS AND
3 5/16 HOLE DIAMETER FLAT WASHERS AND 3 SIZE 9/16 HEXAGONAL NUTS

A: OK

E: LOOSELY FASTEN THE 4 BOLTS

A: HOW?

E: START IN ONE BOLT

A: HOW?

E: PUT THE BOLT THROUGH THE HOLE IN PUMPBASE AND PLATFORM AND
WASHER AND TURN THE NUT CLOCKWISE ONTO THE BOLT AS FAR
AS YOU CAN

A: OK

E: LOOSELY FASTEN THE OTHER 3 BOLTS

A: OK

E: TIGHTEN THE 4 BOLTS

A: HOW?

E: TIGHTEN ONE BOLT

A: HOW?

E: FIND A SIZE 9/16 WRENCH TO FIT THE NUT

A: OK

E: FIND A SIZE 1/2 INCH WRENCH TO FIT THE BOLT HEAD

A: OK

E: USE THE FIRST WRENCH TO HOLD THE NUT FIRMLY

A: OK

E: USE THE SECOND WRENCH TO TURN THE BOLT CLOCKWISE UNTIL FAIRLY TIGHT

A: OK

E: TIGHTEN THE OTHER 3 BOLTS

A: OK

E: INSTALL PUMPBRACE

A: HOW?

E: CONNECT PUMPBRACE TO BELTHOUSINGFRAME

A: HOW?

E: POSITION PUMPBRACE ON BELTHOUSINGFRAME AND POSITION PUMPBRACE ON PUMP

A: OK

E: ATTACH PUMPBRACE TO BELTHOUSINGFRAME

A: HOW?

E: USE SCREWS TO ATTACH PUMPBRACE TO BELTHOUSINGFRAME

A: HOW?

E: GET 2 HALF INCH # 14 SCREWS

A: OK

E: LOOSELY FASTEN THE 2 SCREWS

A: HOW?

E: START IN ONE SCREW

A: HOW?

E: START IN THE SCREW BY TURNING IT CLOCKWISE AS MUCH AS YOU CAN UNTIL
IT HOLDS THERE BY ITSELF

A: OK

E: LOOSELY FASTEN THE OTHER SCREW

A: OK

E: TIGHTEN THE 2 SCREWS

A: HOW?

E: TIGHTEN ONE SCREW

```
A: HOW?

E: FIND A SCREWDRIVER THAT WILL FIT INTO THE NOTCH OF THE SCREW

A: WHY?

E: IN ORDER TO TIGHTEN THE 1 SCREW

A: WHY?

E: IN ORDER TO USE SCREWS TO ATTACH PUMPBRACE TO BELTHOUSINGFRAME

A: WHY?

E: IN ORDER TO ATTACH PUMPBRACE TO BELTHOUSINGFRAME

A: WHY?

E: IN ORDER TO CONNECT PUMPBRACE TO BELTHOUSINGFRAME

A: WHY?

E: IN ORDER TO INSTALL PUMPBRACE

A: WHY?

E: NEXT STEPS ARE TO INSTALL PUMPPULLEY AND INSTALL BELT AND
   INSTALL BELTHOUSINGCOVER

A: WHY?

E: IN ORDER TO TRY TO ACHIEVE ASSEMBLED AIRCOMPRESSOR

A: WHY?

E: ASK THE BOSS

A: OK

E: FIND A SCREWDRIVER THAT WILL FIT INTO THE NOTCH OF THE SCREW

A: OK

E: USE THE SCREWDRIVER TO TURN THE SCREW CLOCKWISE UNTIL TIGHT

A: OK

E: TIGHTEN THE OTHER SCREW

A: OK

E: CONNECT PUMPBRACE TO PUMP

A: OK
```

E: INSTALL PUMPPULLEY

A: HOW?

E: CONNECT PUMPPULLEY TO PUMP

A: HOW?

E: POSITION PUMPPULLEY ON PUMP

A: HOW?

E: PLACE THE PUMPPULLEY ON THE PUMP SHAFT IN SUCH A WAY THAT THE FLAT SIDE OF THE SHAFT IS ALIGNED WITH THE FLAT PART INSIDE THE CENTER HOLE OF THE PUMPPULLEY AND THEN PUSH THE PUMPPULLEY ONTO THE PUMP SHAFT AS FAR AS YOU CAN

A: OK

E: ATTACH PUMPPULLEY TO PUMP

A: HOW?

E: ATTACH THE PUMPPULLEY TO THE SHAFT OF THE PUMP

A: HOW?

E: GET 2 QUARTER INCH # 1/4 SETSCREWS

A: OK

E: LOOSELY FASTEN THE 2 SETSCREWS

A: HOW?

E: START IN ONE SETSCREW

A: HOW?

E: START IN THE SETSCREW BY TURNING IT CLOCKWISE AS MUCH AS YOU CAN UNTIL IT HOLDS THERE BY ITSELF

A: OK

E: LOOSELY FASTEN THE OTHER SETSCREW

A: OK

E: TIGHTEN THE 2 SETSCREWS

A: HOW?

E: TIGHTEN ONE SETSCREW

A: HOW?

E: FIND A SIZE 1/8 ALLENWRENCH

A: OK

E: TURN THE SETSCREW CLOCKWISE UNTIL TIGHT

A: HOW.

E: INSERT THE SHORTER HEXAGONAL END OF THE ALLENWRENCH INTO THE HEXAGONAL HOLE OF THE SETSCREW AND THEN TURN THE SETSCREW CLOCKWISE UNTIL TIGHT

A: OK

E: TIGHTEN THE OTHER SETSCREW

A: OK

E: INSTALL BELT

A: HOW?

E: CONNECT BELT TO MOTORPULLEY AND CONNECT BELT TO PUMPPULLEY

A: HOW?

E: POSITION BELT ON MOTORPULLEY AND POSITION BELT ON PUMPPULLEY

A: HOW?

E: PLACE BELT IN GROOVE OF MOTORPULLEY

A: OK

E: PLACE BELT IN GROOVE OF PUMPPULLEY

A: HOW?

E: IF YOU HAVE TROUBLE GETTING THE BELT AROUND THE PUMPPULLEY TRY MOVING THE MOTOR WHICH MAY MEAN LOOSENING THE FASTENINGS TO ALLOW IT TO SLIDE SIDEWAYS

A: OK

E: ATTACH BELT TO MOTORPULLEY AND ATTACH BELT TO PUMPPULLEY

A: HOW?

```
E: MAKE BELT VERY TIGHT BY ADJUSTING THE POSITION OF THE MOTOR

A: OK

E: INSTALL BELTHOUSINGCOVER

A: OK

E: THANK-YOU

Finished
```

C. Organization of the Report

As mentioned, we have organized work in the project under four headings: problem-solving, natural language, vision, and system integration. Each of these research areas entails two types of work. First there is some "immediate" work that is now being incorporated into the demonstration system. Second there is some longer term research that will have an impact on future systems.

Rather than organizing the report into separate sections corresponding to the four research topics, we have decided to describe in Section II the work in each category that underlies the April 1975 system. This will give the reader an idea of how the present system works and an understanding of its deficiencies. Then in Section III we shall describe the work whose impact will be felt in later systems.

In addition to the front-line research topics, the project needs the help of several important supporting tasks. These are maintenance of and improvements to the QLISP language, maintenance of and improvements to the SRI Artificial Intelligence Center computing system, and development of certain hardware items such as the scanning laser range finder and interface equipment. These supporting tasks will be described in Section IV.

Section V lists the publications and presentations of the CBC project staff during this reporting period.

II. COMPONENTS OF THE APRIL 1975 SYSTEM

A. Introduction

Our 1975 system has already integrated several abilities. It can generate and execute plans for assembly/disassembly at several levels of detail. It can answer queries from the apprentice about the status of the equipment. It can point at parts of the compressor and can name parts pointed to by the apprentice. It has a rudimentary ability for two-way communication using speech. It is the purpose of this section of the report to describe just what underlies each of these abilities.

In Section B we shall discuss our work in generating and executing hierarchical plans. This work has resulted in the development of a concept called the "procedural net for representing hierarchical plans." The basic work on the procedural net has been done by Earl Sacerdoti as part of his doctoral research. In Section C we shall describe the pointing abilities of the system. Underlying pointing is a "geometric model" of the compressor. Work on this model and on pointing is largely the work of Jerry Agin and Georgia Sutherland, with some important early contributions by David Nitzan.

In Section D, a modeling package developed by Richard Fikes is described. This package is used for making deductions on the model in order to answer queries from the apprentice and from other parts of the system. The package also provides a means for updating models of the world during planning. The current system uses some features of this package, and we have plans to use additional features soon.

Our simple system for voice I/O is described in Section E. It consists of a speech synthesizer and a word recognizer--both commercially available.

Finally, in Section F we discuss some of the problems of system integration. Georgia Sutherland has been responsible for this task, and it necessarily has involved her in each of the others also.

B. Reasoning About Assembly/Disassembly Actions

1. Introduction

The assembly and disassembly of equipment is a subtask of virtually all workstation tasks. For example, many troubleshooting jobs and almost all repair jobs require some amount of disassembly and reassembly of the machine. So a major aspect of the consultation task is to model the actions entailed in assembly and disassembly, to compose sequences of these actions to accomplish specific tasks, and to model and monitor the execution of these sequences by a apprentice.

Since the execution of a task is done under a mixed initiative regimen, allowing the apprentice to undertake subtasks in the manner he chooses, we cannot simply build a valid fixed sequence of actions to accomplish a task. Rather, we must include in our model the apprentice's implicit freedom to order the individual steps himself.

In addition to developing a methodology for building and executing hierarchically organized plans of action that reflect this freedom of ordering, we are developing methodologies for defining a hierarchy of relations to describe states of partial assembly, and for defining a hierarchy of actions to alter those states.

In this section we shall first discuss the integrated problem solving and execution monitoring system developed during the last year. Then we shall present the relations and action models that are being used to describe electromechanical equipment, with portions of the semantic model of the air compressor included as an example. Finally, we shall describe how the system develops the dialog presented in Section I.

2. An Integrated Problem Solving and Execution Monitoring System

The NOAH (Nets Of Action Hierarchies) program combines a general purpose problem solver with an execution monitor that is specifically designed for man-machine cooperation. In this section we shall present a simplified explanation of the procedural net, NOAH's representation for actions and plans, of SOUP, the language for giving the system task-specific knowledge, of the planning algorithm, and of the execution algorithm. A complete discussion of the system will appear elsewhere [17].

NOAH is implemented in QLISP [18], and runs as compiled code on a PDP-10 computer under the TENEX time-sharing system.

a. The Procedural Net

The system's plans are built up in a data structure called the procedural net, which has characteristics of both procedural and declarative representations.

Basically, the procedural net is a semantic network of nodes, each of which contains procedural information, declarative information, and pointers to other nodes. Each node represents a particular action at some level of detail. The nodes are linked to form hierarchical descriptions of operations, and to form plans of action.

Nodes at each level of the hierarchy are linked in a partially ordered time sequence by predecessor and successor links. Each such sequence represents a plan at a particular level of detail.

In the current implementation, the nodes are of six types: GOAL nodes represent a goal to be achieved; PHANTOM nodes represent goals that are expected to be true when they are encountered; SPLIT nodes have a single predecessor and multiple successors, and represent a forking of the partial ordering; JOIN nodes have multiple predecessors and a single successor, and represent a rejoining of subplans within the partial ordering; BUILD nodes represent an action that builds up a class of objects; BREAK nodes represent an action that is iteratively applied to the elements of a class of objects.

Each node points to a body of code. The action that the node represents can be simulated by evaluating the body. The evaluation will cause new nodes, representing more detailed actions, to be added to the net. It will also update a hypothesized world model to reflect the effects of the more detailed actions.

Associated with each node is an add list and a delete list. These lists are computed when the node is created. They contain symbolic expressions representing the changes to the world model caused by the action that the node represents.

Thus, the nodes of the procedural net contain two very different representations of an action. The add and delete lists provide a declarative representation of actions that is quite similar to that of STRIPS [19]. The body of code provides a procedural representation similar to that of the new A.I. languages [20]. The declarative representation is used to model the action at the node's own level of detail. The procedural representation is used for generating more detailed subactions at levels of greater detail.

Figure 4 shows the graphic notation used here to display a node of a procedural net.

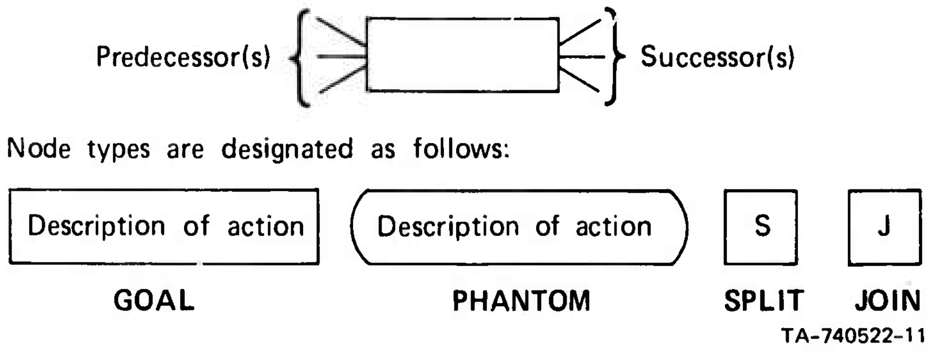


FIGURE 4 GRAPHIC REPRESENTATION OF A NODE

As an example, let us examine a procedural net representing a hierarchy of plans to paint a ceiling and paint a stepladder. The plan can be represented, in an abstract way, as a single node as shown in Figure 5(a). In more detail, the plan is a conjunction, and might be represented as in Figure 5(b). The more detailed subplans to achieve these two goals might be "get paint, get ladder, then apply paint to ceiling," and "get paint, then apply paint to ladder," as depicted in Figure 5(c).

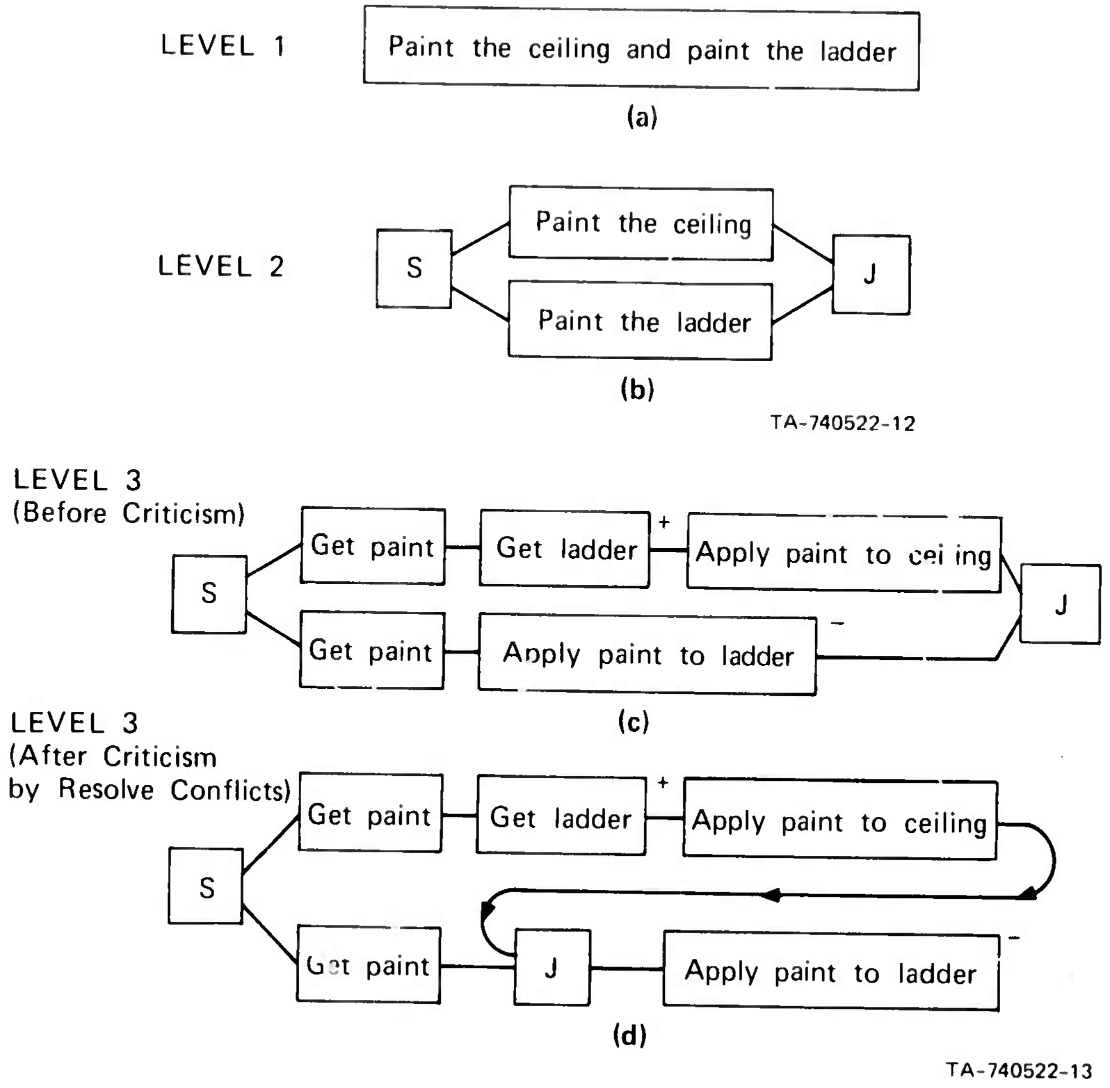


FIGURE 5 PROCEDURAL NET FOR PAINTING

The pictorial representation used here suppresses much of the information associated with each node. The add and delete lists, for instance, are not indicated in the diagrams. They are not hard to infer, however. For example, "get ladder" will cause "has ladder" to be added to the world model, and "apply paint to ceiling" might delete "has paint" from the world model.

Precondition-subgoal relationships are inferred by the system from pointers that indicate which nodes represent detailed expansions of other nodes. These pointers are also omitted in the pictorial representation. The system assumes that every action but the last in such an expansion is a precondition for the last action.

b. Task-Specific Knowledge

Knowledge about the task domain is given to the system in procedural form, written in the SOUP (Semantics Of User's Problem) language. SOUP is an extension of QLISP [18] that is interpreted in an unusual fashion. The process of planning transforms this procedural knowledge into the hybrid procedural net form, which contains both procedural and declarative information, and which represents a hierarchy of solutions to the particular problem at hand.

We will first present the statements of SOUP that have been added to QLISP. (We will call them P-statements.) Then we will describe how the SOUP code is interpreted. Specific examples of SOUP code are presented in Section II.B.3.b.

P-statements that refer to actions are:

PGOAL -

A PGOAL statement is of the form: (PGOAL query pattern APPLY team). Its meaning is similar to the QLISP GOAL. It has an additional argument, the query, that specifies a verbal request for the goal to be ahieved. Evaluation of a PGOAL results in the insertion of a new node in the procedural net. If a true instance of the goal pattern is found in the world model, a PHANTOM node is created. If no true instance is found, a true GOAL node is created.

PBUILD -

A PBUILD statement is of the form: (PBUILD class-name query (ITERATE...)). It specifies an action that will build up a class of objects. The query is a verbal request to build up the class. The iterate statement contains an arbitrary body of code that specifies the subactions entailed in processing one element of the class. Evaluation of a PBUILD statement results in the creation of a BUILD node in the procedural net.

PBREAK -

PBREAK has the same syntax as PBUILD. It specifies an action that iterates through a pre-existing class of objects. Evaluation of a PBREAK statement results in the creation of a BREAK node in the net.

PAND -

A PAND statement is of the form: (PAND exp-1 exp-2 ... exp-n). PAND specifies a collection of expressions that may be evaluated independently. Execution of a PAND statement results in a n-way branching in the procedural net.

P-statements that refer specifically to the world model are:

PIS -

A PIS statement is of the form: (PIS exp). It searches for an instance of the exp that is true in the current world model. If none is found, it causes a failure condition.

PASSERT - A PASSERT statement is of the form: (PASSERT exp). It makes exp be true in the current world model and places exp on the add list of the current node in the procedural net.

PDENY -

PDENY is similar to PASSERT. It makes exp be false and adds exp to the delete list of the current node.

When a P-statement that refers to an action (i.e., PGOAL, PBUILD, PBREAK, or PAND) is evaluated, it does not cause an arbitrarily deep computation, as would most PLANNER-like languages [20]. Rather, the action is mocked (we mean simulated, but we're already using that term for another operation) and the world model is updated as if the deep computation had been done and the action were accomplished in full detail. The information necessary to continue the computation to further depths is stored as the body of code associated with the new nodes that are created in the procedural net. For example, when a PGOAL statement is evaluated (thus mocking the action that achieves the goal), the team of functions associated with the statement is placed as the body of the new node representing that goal. When a PBUILD or PBREAK statement is evaluated, the ITERATE clause is stored as the new node's body. This type of evaluation results in the creation of hierarchies of plans of increasing detail. This scheme thus extends the ability to do hierarchical planning as was done by ABSTRIPS [21] from a syntactically oriented declarative representation to SOUP's semantically oriented procedural representation.

c. The Planning Algorithm

Initially, NOAH is given a goal to achieve.

NOAH first builds a procedural net that consists of a single goal node to achieve the given goal. This node has a list of all relevant SOUP functions as its body, and it represents the plan to achieve the goal at a very high level of abstraction. This one-step plan may then be expanded by the planning algorithm.

The planning algorithm of the NOAH system is simple. Its input is a procedural net. It expands the most detailed plan in the net by simulating each node of the plan in turn. In addition to building a detailed model of the effects of each action in the plan, the simulation of each node will produce child nodes. Thus by simulating the plan, a new, more detailed plan will be created.

The individual subplan for each node will be correct, but there is as yet no guarantee that the new plan, taken as a whole, will be correct. There may be interactions between the new, detailed steps that render the overall plan invalid. For example, the individual expansions entailed in generating the plan in Figure 5(c) from that in Figure 5(b) are correct, yet the overall plan is invalid, since it permits painting the ladder before painting the ceiling.

Before the new detailed plan is presumed to work, the planning system must take an overall look at it to ensure that the local expansions make global sense. This global examination is provided by a group of critics. These critics serve a purpose somewhat similar to that of the critics of Sussman's HACKER [22], except that for NOAH they are constructive critics, designed to add constraints to as yet unconstrained plans, whereas for HACKER they were destructive critics whose purpose was to reject incorrect assumptions reflected in the plans.

For example, a constructive critic will alter the painting plan in Figure 5(c) to ensure that the endangered subgoal, painting the ceiling, is achieved before the step that endangers it, namely painting the ladder. After this critic has altered the plan, it will appear as in Figure 5(d). Note that planners that use a linear representation of plans cannot solve such problems without extensive use of backtracking, or sophisticated plan optimization.

The algorithm for the planning process, then, is as follows:

(1) Simulate the most detailed plan in the procedural net. This will have the effect of producing a new, more detailed plan.

(2) Criticize the new plan, performing any necessary reordering or elimination of redundant operations.

(3) Go to Step 1.

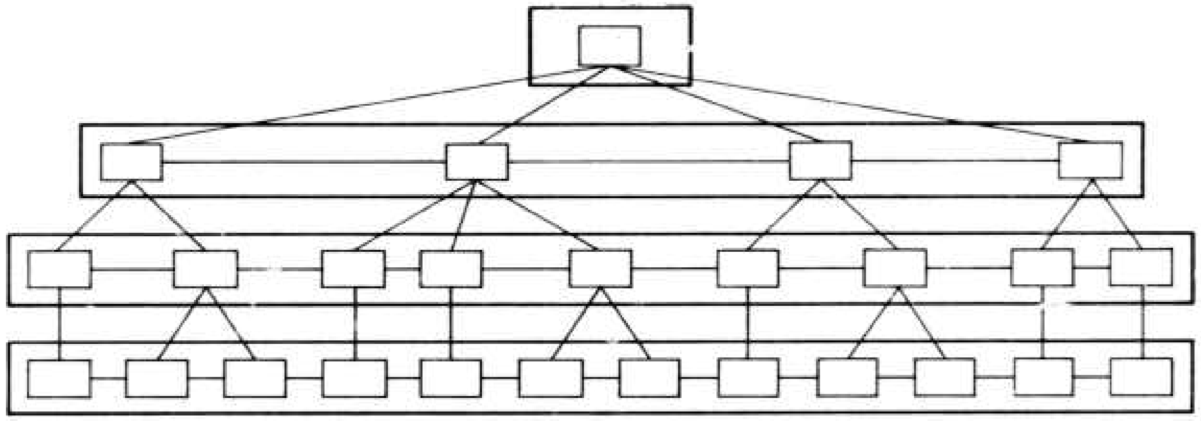
Clearly, this algorithm is an oversimplification, but for the purposes of this report we may imagine that the planning process continues until no new details are uncovered. (In fact, for the complete problem solving and execution monitoring system, a local decision must be made at every node about whether it should be expanded.)

A detailed example of the generation of a plan for assembling a disassembled air compressor is presented in Section II.B.4.

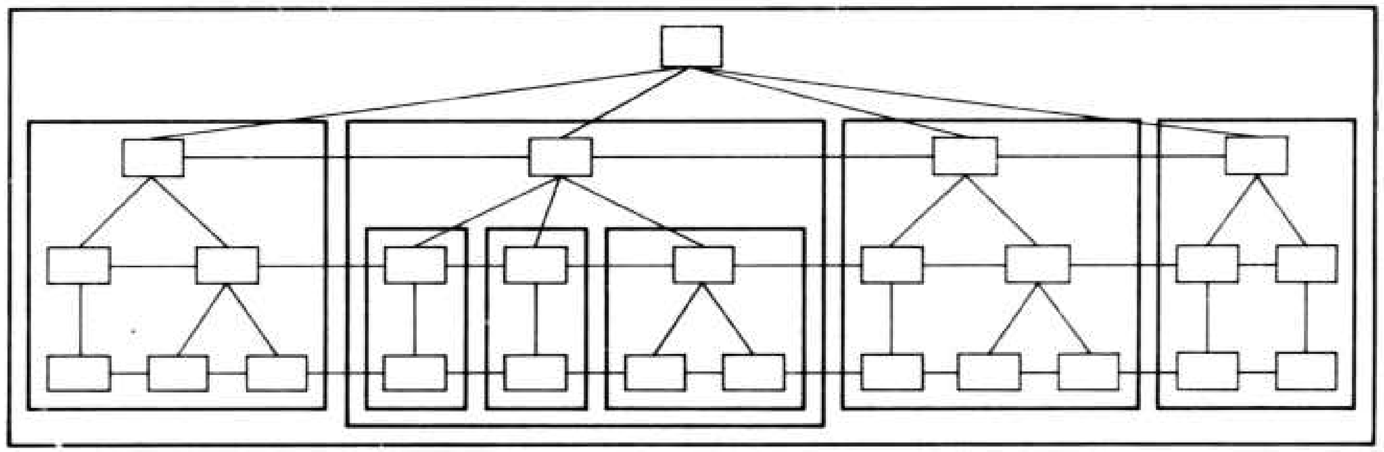
A recent paper [23] discusses in more detail the problem solving algorithms, the constructive critics, and a comparison of this approach with other recent work.

d. The Execution Algorithm

The output of the planning process is a procedural net, which was developed as a hierarchy of partially linearized plans. Figure 6(a) suggests the planner's viewpoint of the procedural net. The same procedural net is also the input to the execution portion of the system. The execution algorithm views the net differently, however. It sees the procedural net as a collection of action hierarchies, as suggested by Figure 6(b). An action



(a) THE PROCEDURAL NET FROM THE PLANNER'S POINT OF VIEW



(b) THE PROCEDURAL NET FROM THE EXECUTION MONITOR'S POINT OF VIEW

SA-3805-6

FIGURE 6 DIFFERENT PERSPECTIVES OF A PROCEDURAL NET

hierarchy, consisting of a node representing an action, together with its children nodes representing subactions, together with their descendent nodes, will be termed a wedge. The execution monitor views the procedural net as a single wedge, to which it applies the following algorithm:

(1) Ask the apprentice to accomplish the action represented by the node at the top of the wedge. This is done by saying the node's query to him.

(2) If he responds positively, assume the current wedge has been accomplished, and so assume the current wedge has been successfuly executed.

(3) If he responds negatively, assume he needs a more detailed breakdown of the action, and so execute in turn all the subwedges headed by children of the current wedge.

Actually, the algorithm is more complicated than this. The apprentice may make a wider variety of responses. To each query, his possible responses are:

Affirmative responses - Yes, OK, Yup, Check...

This type of response indicates that the apprentice understands the instruction and is able to do it. In fact, in the current implementation, the CBC program assumes that the task has been completed. This type of response signals the execution algorithm to move on to the succeeding wedge.

Negative responses - No, How, Can't...

This type of response indicates that the apprentice needs help before he can perform the indicated action. This signals the procedural net program to move to the first child node.

Repeat responses - what, Huh...

This type of response indicates that the apprentice did not hear the instruction or was not sure what was said. Please repeat. The program merely repeats the query.

Motivation response - why

The apprentice wants to know why a certain task needs to be done. The current response to this question is for the procedural net program to list the tasks that remain to be done at that level. These correspond to the queries of successive nodes that have the same parent as the current node. If the apprentice still wants to know why, the program then repeats the query associated with the parent node. This process may be repeated until there are no more parent nodes.

Escape responses - Break, Pause, Graphics...

These responses are temporary expedients to allow interaction with other portions of the CBC system, such as the graphics package. (See the description of the CBC Expert System in Section II.F below). This allows the apprentice the opportunity to ask questions like "Where," "What is this," and the like.

When the top wedge of the procedural net has been successfully executed, the Execution Phase terminates with "Thank You".

The current system's ability to monitor the execution of a task is limited, and its ability to respond intelligently to unexpected failures is nil. During the coming year we plan to focus considerably more energy on the execution monitoring aspect of the assembly/disassembly task.

3. The Semantics of Assembly and Disassembly Actions

a. Relations

In order to model the processes of assembling and disassembling equipment, we have defined a set of entities and relations that are used to describe a device in arbitrary stages of assembly and disassembly. The definitions being used in the current system are given in this section along with examples that relate to the air compressor. These definitions form the basis for the SOUP functions that produce a procedural net of plans for assembly and disassembly. The SOUP functions will be described in Section II.B.3.b.

1) Connections

A "connection" is defined between any

two components that are fastened to each other when a device is fully assembled. The elements of a connection are an ordered pair of components, where the ordering implies that the first component is fastened to the second component. For example, we have (CONNECTION PUMP PLATFORM) rather than (CONNECTION PLATFORM PUMP), since one would say "Connect the pump to the platform" rather than "Connect the platform to the pump". In the cases where no intuitive or actual assymetry exists between the components, we determine whether one of the components supports the other in the fully assembled device. If so, the supportee is the first element of the pair. In the remaining cases, where neither criterion applies, an arbitrary ordering of the components is chosen.

The connection relations are used for defining the canonical locations of components for the fully assembled device, rather than for specifying actual locations of components at any particular time. Thus, connection relations remain unchanged during assembly and disassembly.

ii) Fastenings

Connections usually represent components that are fastened together by bolts, screws, machine screws, or setscrews. The objects that fasten components together are referred to as "fastenings". A bolt with its nut and washers is a standard type of fastening. Examples of fastenings include each of the four nut-washer-bolt assemblies that connect the pump to the platform and each of the ten sheet metal screws that fasten the belt housing cover to the frame.

The association between a fastening and the components it fastens is represented by storing with each connection the fastenings that hold it together. Table 1 lists some of the connections and fastenings that are currently used by the CBC system

---

Table 1: CONNECTION Expressions for the Air Compressor

```
(CONNECTION Pump Platform)
        FASTENFN=BOLTON
        FASTENING=(_BOLT Pumpbase Platform _WASHER _NUT)
                LENGTH=1.25
                DIAMETER=.3125
                NUMBER=4

(CONNECTION Motor Platform)
        FASTENFN=BOLTON
        FASTENING=(_BOLT Motorbase Platform _WASHER _NUT)
                LENGTH=1
                DIAMETER =.25
                NUMBER=4
```

```
(CONNECTION Pumppulley Pump)
        FASTENFN=WHEELON
        FASTENING=(_SETSCREW Pumppulley Pumpshaft)
                LENGTH=.25
                DIAMETER=.25
                NUMBER=2

(CONNECTION Motorpulley Motor)
        FASTENFN=WHEELON
        FASTENING=(_SETSCREW Motorpulley Motorshaft)
                LENGTH=.25
                DIAMETER=.25
                NUMBER=1

(CONNECTION Belt Pumppulley)
        FASTENFN=BELTON

(CONNECTION Belt Motorpulley)
        FASTENFN=BELTON

(CONNECTION Pumpbrace Pump)
        FASTENFN=BOLTON
        FASTENING=(_BOLT Pumpbrace Pumptop)
                LENGTH=.75
                DIAMETER=.3125
                NUMBER=1
```

```
(CONNECTION Pumpbrace Belthousingframe)

        FASTENFN=SCREWON

        FASTENING=(_SCREW Pumpbrace Belthousingframe)

                LENGTH=.5

                DIAMETER=.25

                NUMBER=2


(CONNECTION Belthousingcover Belthousingframe)

        FASTENFN=SCREWON

        FASTENING=(_SCREW Belthousingcover Belthousingframe)

                LENGTH=.5

                DIAMETER=.25

                NUMBER=10


(CONNECTION Aftercooler Pump)

        FASTENFN=SCREWINTO

                FEMALE=Pump

                MALE=Aftercooler


(CONNECTION Aftercooler Aftercoolerelbow)

        FASTENFN=SCREWINTO

                FEMALE=Aftercooler

                MALE=Aftercoolerelbow


--------------------------------------------------------------------
```

iii) Chunks

Two components X and Y have a connection path defined between them if there exists some sequence of connections between components X-C1, C1-C2, ..., CN-Y. A "Chunk" is a collection of components that have connection paths defined among them and are all positioned with respect to each other; i.e., a positioned subassembly. We define the "loose ends" of a chunk to be that collection of connection relations between members of the chunk and nonmembers of the chunk. For example, an isolated single component can be considered a chunk and all the component's connections would be considered loose ends. If a chunk is positioned with respect to some other chunk, then the loose ends set of the newly formed (larger) chunk consists of those connections that were elements of exactly one of the loose ends sets of the two old chunks (since connections that were loose ends of both the old chunks are now connections between members of the new chunk.)

iv) Fastened/Unfastened

If X is a fastening, then the relation (FASTENED X) is true whenever X has been inserted in its proper place and tightened. For example, a fastened nut and bolt assembly means that the bolt, nut, and all associated washers are firmly in place. Similarly, (UNFASTENED X) is true when the fastening X is disassembled and removed from its proper position.

v) Positioned/Extracted

If components X and Y have a connection defined between them, then the relation (POSITIONED X Y) is true whenever X and Y are in the same chunk. Similarly, if components X and Y have a connection defined between them, then the relation (EXTRACTED X Y) is true whenever X and Y are in different chunks and X and Y do not restrict each other's movement in any meaningful way.

Chunk descriptions and the relations POSITIONED and EXTRACTED are redundant, since being a loose end implies EXTRACTED and not being a loose end implies POSITIONED.

It is convenient to define the relational form (POSITIONED X) to be true for a component X if and only if (POSITIONED X Y) is true for each Y such that (CONNECTION X Y) is defined. Similarly, (EXTRACTED X) is defined to be true for a component X if and only if (EXTRACTED X Y) is true for each Y such that (CONNECTION X Y) is defined.

vi) Attached/Detached

If (CONNECTION X Y) is defined, then

the relation (ATTACHED X Y) is true whenever (FASTENED Z) is true for all fastenings Z associated with the X-Y connection. (Note that ATTACHED and FASTENED can only be true if POSITIONED is also true.) Similarly, if (CONNECTION X Y) is defined, then the relation (DETACHED X Y) is true whenever (UNFASTENED Z) is true for all fastenings Z associated with the X-Y connection.

We also define the relational form (ATTACHED X) to be true for a component X if and only if (ATTACHED X Y) is true for each Y such that (CONNECTION X Y) is defined. Similarly, (DETACHED X) is defined to be true for a component X if and only if (DETACHED X Y) is true for each Y such that (CONNECTION X Y) is defined.

vii) Connected/Disconnected

For any components X and Y, the relation (CONNECTED X Y) is true whenever (ATTACHED X Y) is true, and the relation (DISCONNECTED X Y) is true whenever (EXTRACTED X Y) is true. (Note that ATTACHED implies POSITIONED and EXTRACTED implies DETACHED.)

We also define the relational form (CONNECTED X) to be true for a component X if and only if (CONNECTED X Y) is true for each Y such that (CONNECTION X Y) is defined. Similarly, (DISCONNECTED X) is defined to be true for a component X if and only if (DISCONNECTED X Y) is true for each Y such that (CONNECTION X Y) is defined.

viii) Installed/Removed

For any two components X and Y of the same device, the relation (INSTALLED X Y) is true whenever X and Y are in the same chunk and when (CONNECTED X Z) is true for each Z (in the chunk with X) for which (CONNECTION X Z) is defined. Also, if X and Y are components of the same device, then the relation (REMOVED X Y) is true whenever X and Y are in separate chunks and X and Y do not restrict each other's movement in any meaningful way.

We also define the relational form (INSTALLED X) to be true for a component X if and only if (INSTALLED X Y) is true for each Y such that (CONNECTION X Y) is defined. Similarly, (REMOVED X) is defined to be true for a component X if and only if (REMOVED X Y) is true for each Y such that (CONNECTION X Y) is defined.

ix) Assembled/Disassembled

For any device X, the relation (ASSEMBLED X) is true whenever (INSTALLED Y) is true for each component Y of X. Similarly, for any device X, the relation (DISASSEMBLED X) is true whenever (REMOVED Y) is true for each component Y of X.

x) Onhand

If X is a component of a fastening or is a tool, then the relation (ONHAND <object-type of X> X) is true whenever X is in use by or immediately available to the technician. This relation is typically used for objects such as screws, bolts, nuts, washers, setscrews, wrenches, or screwdrivers.

xi) Startedin/Takenout

If X is a fastening, then (STARTEDIN X) is true whenever the components of X have been positioned with respect to the connection that they fasten. Similarly, if X is a fastening, then (TAKENOUT X) is true whenever the components of X have been removed from the connection that they fasten.

xii) Tightened/Loosened

If X is a fastening, then (TIGHTENED X) is true whenever the components of X have been tightened. For example, a bolt-washer-nut fastening is tightened by turning the nut onto the bolt until tight. Similarly, (LOOSENED X) is true whenever the components of a fastening are loosened. The relations STARTEDIN/TAKENOUT and TIGHTENED/LOOSENED for fastenings are analogous to the relations POSITIONED/EXTRACTED and CONNECTED/DISCONNECTED for components.

xiii) Spindled

SPINDLED is a relation that applies to a fastening and a list of components. The SPINDLED relation, if true, implies that the objects are fastened together in the order listed in the relation. An example is (SPINDLED Bolt1 Pump Platform Washer1 Nut1). This type of relation is used to keep track of associations between particular bolts, nuts, washers, and the like.

xiv) Applied

The relation (APPLIED <tool-type> <fastening-name>) is true whenever a tool of the specified type is being used by the apprentice to tighten or loosen the specified fastening.

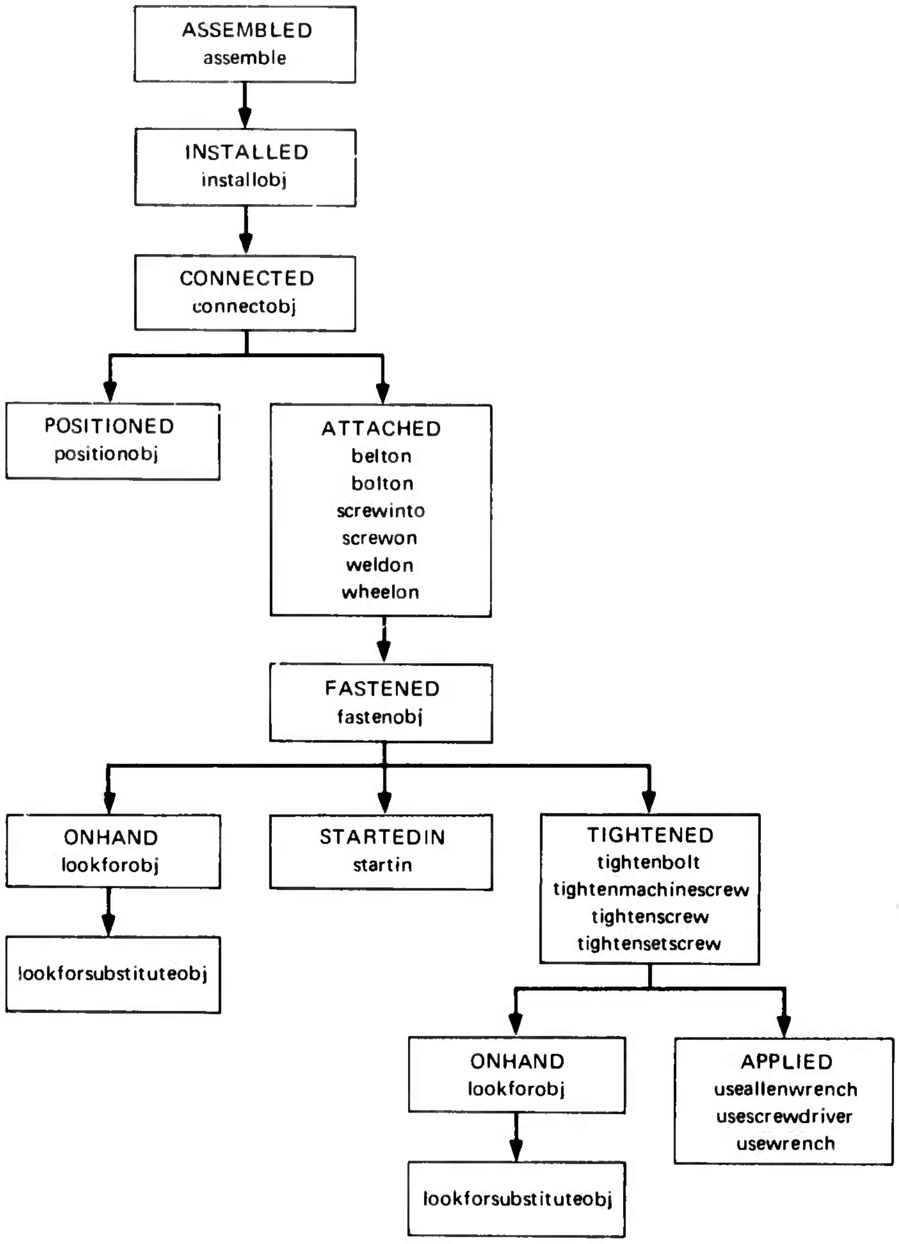
b. Describing Actions for Assembly and Disassembly

The relations described in the preceeding section imply a hierarchy of actions to be used in forming plans about assembly and disassembly of equipment. Accordingly, SOUP functions have been written to achieve all the relations, referring explicitly to tools and fastenings when appropriate.

Figures 7 and 8 show the relations/actions hierarchy for assembly and disassembly planning. The upper case names refer to relations; the actions, or SOUP functions corresponding to those relations, are lower case names. The vertical lines connecting the blocks in the figures indicate different levels of the hierarchy, and imply function calls. When a SOUP function calls more than one lower level SOUP function, this is indicated by a horizontal line. Some of the assembly actions are described below.

```
ASSEMBLE (QLAMBDA (ASSEMBLED ←OBJ)
         (APPLY (FUNCTION PAND)
                (MAPCAR (GETP $OBJ (QUOTE COMPONENTS))
                     (FUNCTION (LAMBDA (X)
                          (↑(PGOAL (INSTALL (@ X))
                                   (INSTALLED (@ X))
                                   APPLY (INSTALLOBJ)))))
```

A list of COMPONENTS has been previously set up. To assemble the object, set up a goal for installing each component separately. If any component is already installed, this information will be already stored in the world model, PGOAL will discover this fact, a PHANTOM node will be inserted in the net, and the apprentice will never be instructed to install a component unnecessarily.



SA-3805-7

FIGURE 7   FLOW CHART FOR SOUP FUNCTIONS IN THE ASSEMBLY DOMAIN

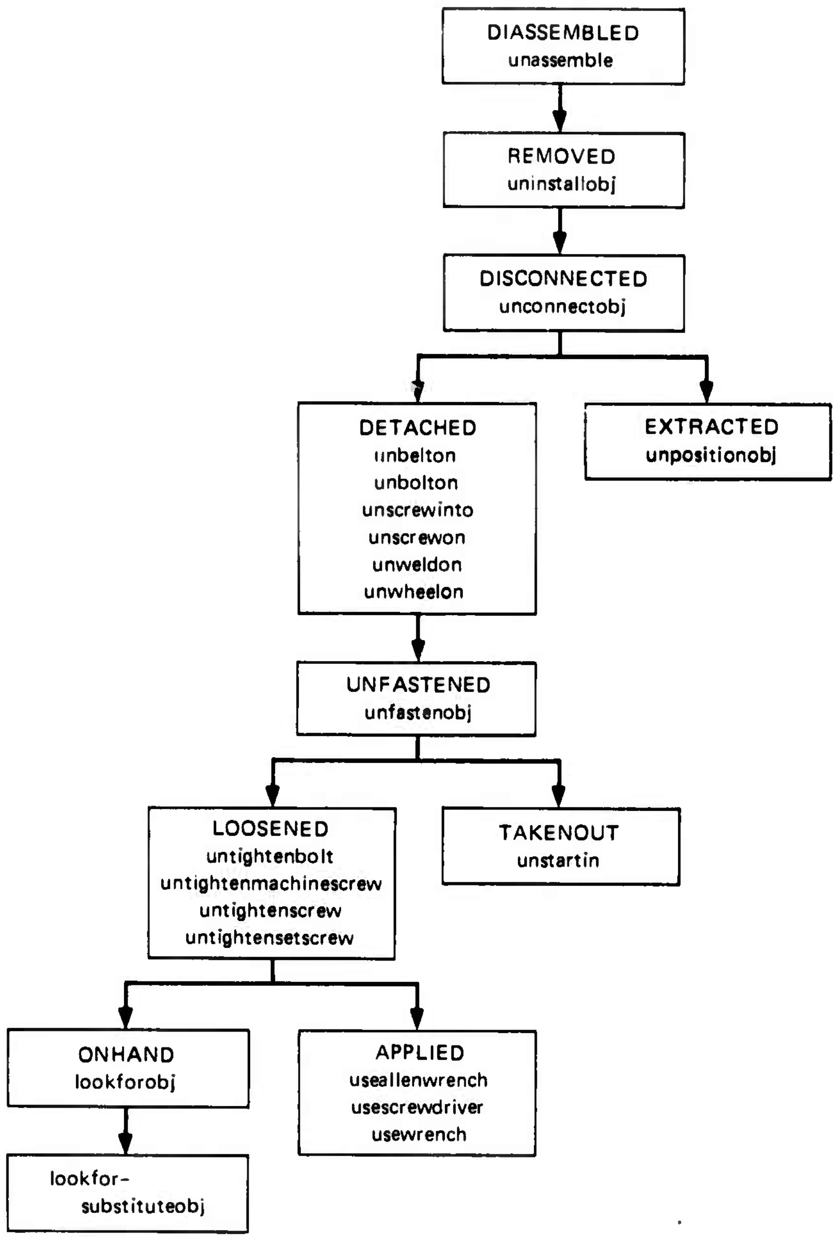


FIGURE 8 FLOW CHART FOR SOUP FUNCTIONS IN THE DISASSEMBLY DOMAIN

```
INSTALLOBJ (QLAMBDA (INSTALLED ←OBJ)
   (QPROG (←OBJ2)
        (APPLY (FUNCTION PAND)
              (MAPCAR (CDR (INSTANCES
                       (CONNECTION $OBJ ←OBJ2)))
                       (FUNCTION (LAMBDA (X)
                          (SETQ X (CADDR X))
                          (↑(PGOAL (CONNECT $OBJ TO (@ X))
                                  (CONNECTED $OBJ (@ X))
                                  APPLY (CONNECTOBJ)))]
```

Each defined connection relation is explicitly stored in the world model. The QLISP function, INSTANCES, will retrieve all those connections that involve $OBJ. Then a goal is set up to connect $OBJ to all other components. Notice that the P-function, PAND, is used to make a conjunction of all the connections; the procedural net critiques will put the conjuncts in the best order for execution.

```
CONNECTOBJ (QLAMBDA (CONNECTED ←OBJ1 ←OBJ2)
        (PGOAL (POSITION $OBJ1 ON $OBJ2)
              (POSITIONED $OBJ1 $OBJ2)
              APPLY (POSITIONOBJ))
        (PGOAL (ATTACH $OBJ1 TO $OBJ2)
              (ATTACHED $OBJ1 $OBJ2)
              APPLY ((@(QGET (CONNECTION $OBJ1 $OBJ2) FASTENFN))))]
```

In order to connect two objects, they must first be positioned, and then attached to each other. The type of attachment is implied by the fastening function property, FASTENFN, stored on each connection relation.

```
POSITIONOBJ (QLAMBDA (POSITIONED ←OBJ1 ←OBJ2)
   (QPROG (←MOBJ ←FOBJ)
        (SELECTQ (QGET (CONNECTION $OBJ1 $OBJ2) FASTENFN)
                (WHEELON (PGOAL (PLACE THE $OBJ1 ON THE $OBJ2
                                SHAFT IN SUCH A WAY THAT THE
                                FLAT SIDE OF THE SHAFT IS ALIGNED
                                WITH THE FLAT  PART INSIDE THE
                                CENTER HOLE OF THE $OBJ1 AND THEN
                                PUSH THE $OBJ1 ONTO THE $OBJ2
                                SHAFT AS FAR AS YOU CAN)
                                (POSITIONED $OBJ1 $OBJ2)
                                APPLY NIL))
                (BELTON (PGOAL  (POSITION $OBJ1 IN GROOVE OF $OBJ2)
                                (POSITIONED $OBJ1 $OBJ2)
                                APPLY (EXPLAINBELTPOSITION)))
                (SCREWINTO (QGET (CONNECTION $OBJ1 $OBJ2)
                                 FEMALE ←FOBJ MALE ←MOBJ)
                           (PGOAL (PLACE THE END OF THE $MOBJ
                                NEXT TO THE HOLE OF THE $FOBJ)
                                (POSITIONED $OBJ1 $OBJ2)
                                APPLY NIL))
```

```
                    (PGOAL (PLACE THE $OBJ1 AND THE $OBJ2 NEXT TO EACH
                                  OTHER IN SUCH A WAY THAT HOLES IN THE $OBJ1
                                  ARE MATCHED UP WIH HOLES IN THE $OBJ2)
                                  APPLY NIL))))]
```

This is a general positioning function that gives instructions about four different types of positionings: wheels, belts, components that screw into other components, and, finally, components that are connected with fastenings such as bolts, or screws.

```
BOLTON (QLAMBDA (ATTACHED ←OBJ1 ←OBJ2)
        (PGOAL (USE BOLTS TO CONNECT $OBJ1 AND $OBJ2)
               (FASTENED BOLT ($OBJ1 $OBJ2))
               APPLY (FASTENOBJ))]
```

This is an example of an attachment function, applying to components that are to be bolted together.

```
FASTENOBJ (QLAMBDA (FASTENED ←FASTENING (←OBJ1 ←OBJ2))
   (QPROG (←OBJECTS ←LENGTH ←N ←DIAMETER ←C
                 ←COMPUTATION ←VARIABLES
                   (←FASTENINGS (MKATOM (CONCAT $FASTENING "S"))))
       (QGET (CONNECTION $OBJ1 $OBJ2)
                FASTENING ←OBJECTS   LENGTH ←LENGTH
                 NUMBER ←N SHAPE ←SHAPE   DIAMETER ←DIAMETER)
       (COND ((EQ $N 1)(MATCHQ ←FASTENINGS $FASTENING)))
```

```
(We have omitted a section of code here that sets $VARIABLES
 equal to the list of variables in $OBJECTS.  These
 are the bolts, washers, nuts, screws, etc., which
 are not explicitly specified by name.  The code
 constructs a query of the form "get two #3 bolts and
 two #4 washers and two #3 nuts" to be used in the
 subsequent PBUILD statement.  This special query is
 saved as the value of $COMPUTATION.)
(PBUILD (CLASS ←←C)
        (GET (! (@ (CDR (APPLY (FUNCTION APPEND)
                                    $COMPUTATION)))))
        (ITERATE $N $VARIABLES
           (MAP2C $COMPUTATION $VARIABLES
                (FUNCTION (LAMBDA (X Y)
                   (PGOAL (GET ONE (! (@ (CDDR X))))
                          (ONHAND (@ (MKCONSTANT Y)) (@ Y))
                          APPLY (LOOKFOROBJ) TYPE (@ (CDDR X)))))
(QPUT (CONNECTION $OBJ1 $OBJ2) HARDWARE $C)
(PBREAK (CLASS ←←C)
        (LOOSELY FASTEN THE $N $FASTENINGS)
        (ITERATE $N $VARIABLES
                (PGOAL (START IN ONE $FASTENING)
                       (STARTEDIN $FASTENING (@(↑ $VARIABLES)))
                       APPLY (STARTIN))
                (MAPC $VARIABLES (FUNCTION (LAMBDA (V)
                       (SETQ V (LIST (QUOTE ONHAND)
                                     (MKCONSTANT V) V))
                       (PDENY (@ (↑ (@ V)))))))))
        (PBREAK (CLASS ←←C)
                (TIGHTEN THE $N $FASTENINGS)
                (ITERATE $N $VARIABLES
                        (PGOAL (TIGHTEN ONE $FASTENING )
                               (TIGHTENED $FASTENING (@(↑ $VARIABLES)))
                               APPLY (TIGHTENBOLT
                                      TIGHTENMACHINESCREW
                                      TIGHTENSCREW
                                      TIGHTENSETSCREW)))))
```

In fastening objects together, first get the fastenings, then loosely fasten them, then tighten them. The initial QGET statement retrieves the properties of the fastenings themselves (length, diameter, number, and so on). The PBUILD statement instructs the apprentice to physically get the fastenings. The two PBREAK statements tell the apprentice what to do with the fastenings to make the connection.

```
TIGHTENBOLT (QLAMBDA (TIGHTENED BOLT (←BOLT ←←O))
   (QPROG (←WRENCH1 ←WRENCH2 ←NUT)
        (COND ((OR (NULL $O)
                   (NOT (EQ (GETP (MATCHQ ←NUT (FLAST $O)) (QUOTE TYPE))
                            (QUOTE NUT)))))
                (PGOAL (FIND A WRENCH TO FIT THE BOLT HEAD)
                       (ONHAND WRENCH ←WRENCH2)
                       APPLY (LOOKFOROBJ) TYPE WRENCH)
                (PDENY (ONHAND WRENCH $WRENCH2))
```

```
            (PGOAL (USE THE WRENCH TO TURN THE BOLT
                        CLOCKWISE UNTIL FAIRLY TIGHT)
                       (APPLIED $WRENCH2 $BOLT)
                       APPLY (USEWRENCH)))
          (T (PGOAL (FIND A WRENCH TO FIT THE NUT)
                       (ONHAND WRENCH ←WRENCH1)
                       APPLY (LOOKFOROBJ) TYPE WRENCH)
             (PDENY (ONHAND WRENCH $WRENCH1))
             (PGOAL (FIND A WRENCH TO FIT THE BOLT HEAD)
                       (ONHAND WRENCH ←WRENCH2)
                       APPLY (LOOKFOROBJ) TYPE WRENCH)
             (PDENY (ONHAND WRENCH $WRENCH2))
             (PGOAL (USE THE FIRST WRENCH TO HOLD THE NUT FIRMLY)
                       (APPLIED $WRENCH1 $NUT)
                       APPLY (USEWRENCH))
             (PGOAL (USE THE SECOND WRENCH TO TURN THE BOLT
                        CLOCKWISE UNTIL FAIRLY TIGHT)
                       (APPLIED $WRENCH2 $BOLT)
                       APPLY (USEWRENCH))
             (PASSERT (ONHAND WRENCH $WRENCH1))))
     (PASSERT (ONHAND WRENCH $WRENCH2))]
```

This is an example of a particular tightening function. (TIGHTENSCREW, TIGHTENMACHINESCREW, and TIGHTENSETSCREW are similar.) The process requires the apprentice to get the tools and use them. While the tools are in use, they are declared to be "unavailable" by the PDENY statements. They are later made available again by PASSERT.

It is important to notice that none of the SOUP-functions described in this section actually refer to an air compressor in any way. They refer to members of a COMPONENTS list and to a set of CONNECTION relations that are stored in the world model. Tools are explicitly referred to in the lower level functions, as are fastenings such as screws, nuts, and bolts.

All the specific information about the air compressor is coded into a single initializing function that sets up the world model before any of the procedural net is built. The initialization function contains many expressions of the following form:

```
(ASSERT (CONNECTION PUMP PLATFORM)
         FASTENFN BOLTON
         FASTENING (←BOLT PUMPBASE PLATFORM ←WASHER ←NUT)
         LENGTH 1.25
         DIAMETER .3125
        NUMBER 4)
```

(Other examples were shown in Table 1 above.)

Thus we expect that it would be relatively easy to convert the current planning and execution monitoring program to another example of electromechanical equipment.

4. Examples from a Demonstration System

a. A Detailed Trace of the Planning Algorithm

The NOAH system is invoked with a top-level goal as its argument. An initial procedural net is built, with the structure shown in Figure 9(a). The body of the GOAL node is the value of a QLISP variable that is bound to a list of all top-level SOUP functions (in this case, ASSEMBLE and DISASSEMBLE).

The planning algorithm is then applied to the most detailed plan in the initial net. At this point, the most detailed plan consists of the single GOAL node. The GOAL node is simulated, which means that the functions in its body, namely ASSEMBLE and DISASSEMBLE, are applied in turn to its pattern (ASSEMBLED AIRCOMPRESSOR) until some function does not fail. (Note that this is essentially equivalent to evaluating the QLISP statement (GOAL (ASSEMBLED AIRCOMPRESSOR) APPLY (ASSEMBLE DISASSEMBLE)).)

The function ASSEMBLE was reproduced in

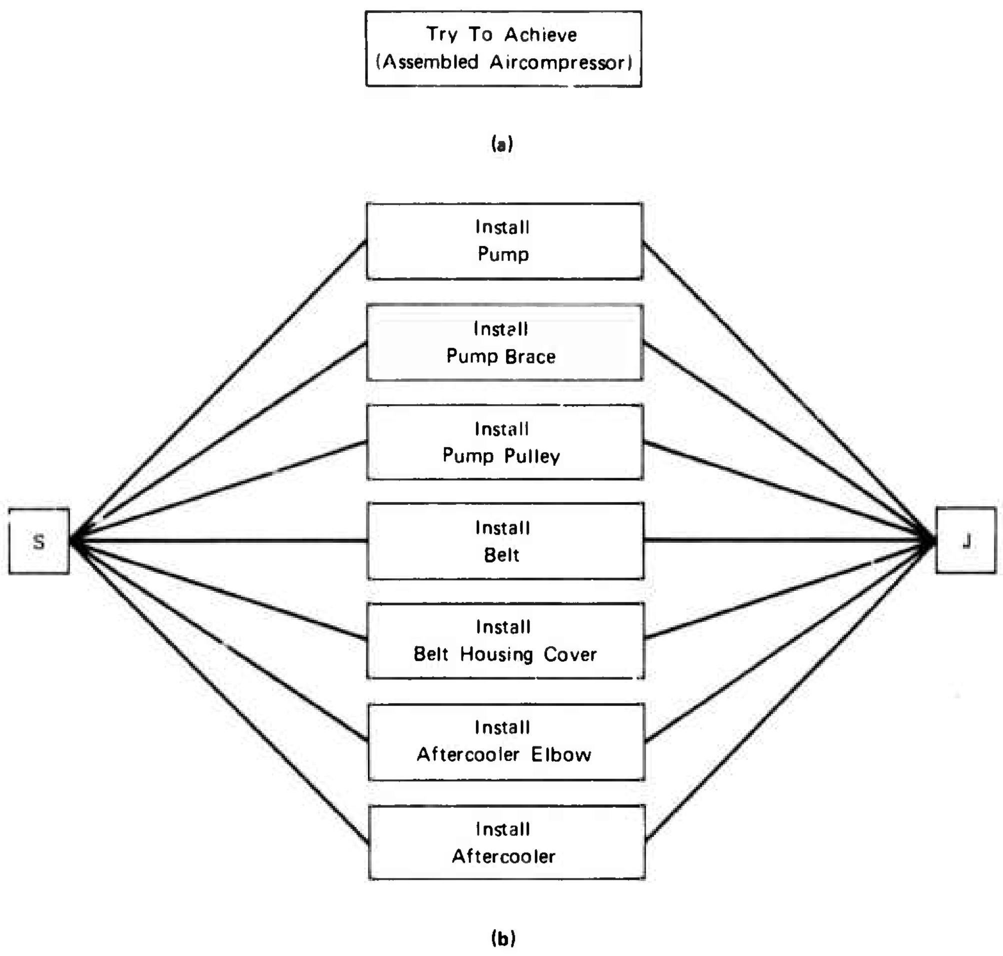


FIGURE 9 PROCEDURAL NET OF THE DEMONSTRATION SYSTEM

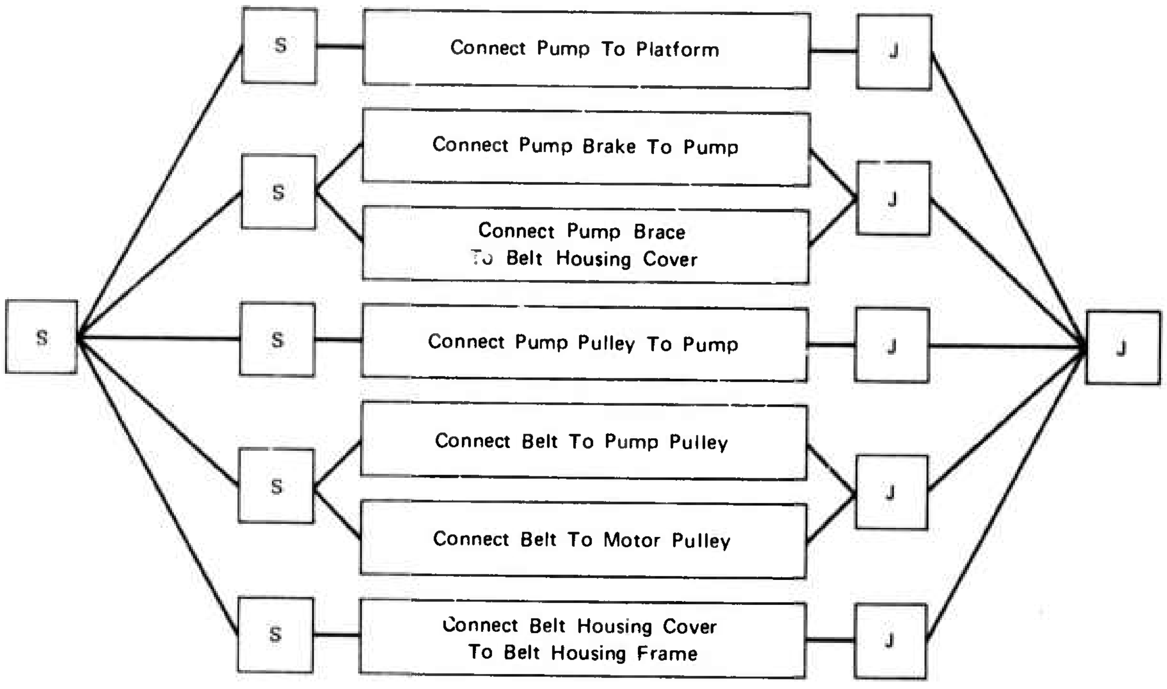
Section 3 above. It evaluates the expression (PAND (PGOAL (Install COMP-1) (INSTALLED COMP-1) APPLY (INSTALLOBJ)) ... (PGOAL (Install COMP-n)(INSTALLED COMP-n) APPLY (INSTALLOBJ))), for components COMP-1 through COMP-n of the air compressor. Evaluation of the PAND statement results in the evaluation of each of the PGOAL statements. When each PGOAL statement is evaluated, the expression (INSTALLED COMP-1) is asserted in the current world model, and a new node is placed in the procedural net. The new node will have (INSTALLED COMP-1) as its pattern, and will have (INSTALLOBJ) as its body.

Because the PGOALs are inside a PAND statement, the nodes that they generate in the procedural net are not linked linearly. Rather, they are linked in parallel to special SPLIT and JOIN nodes, as depicted in Figure 9(b). This completes the simulation of this level of the plan.

At this level the critics do not yet have a sufficiently detailed model to analyze, and so they propose no alterations to the plan.

Next the new, more detailed plan is simulated. This means that each of its nodes, in turn, will be simulated. First, the SPLIT node is simulated, and this results in the creation of a new SPLIT node at a lower level of the procedural net. Then, each of the GOAL nodes is simulated, by applying the function INSTALLOBJ to its pattern. INSTALLOBJ contains a PAND statement (see Section 3 above). Each application of INSTALLOBJ represents the installation of a particular component. Evaluation of the PAND statement in INSTALLOBJ will result in the generation of a parallel group of GOAL nodes specifying all the connections to be made to the given component in the fully assembled air compressor. Finally, the JOIN node is simulated, resulting in the creation of a new JOIN node. The new, most detailed level of the procedural net looks like Figure 9(c) at this point. Now, if preconditions had been defined for each connection, the critics would be able to modify this structure to appear as in Figure 9(d). However, the preconditions have not as yet been encoded, and so the demonstration system relies on the intelligent initial ordering of the components list by the definer of the domain. The critics do, however, clean up any superfluous SPLITs and JOINs.

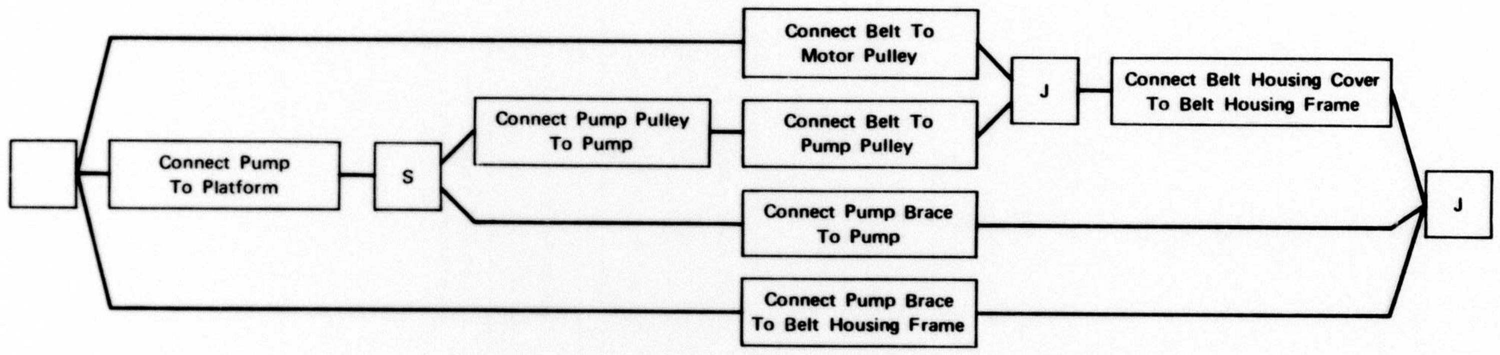
Expansion of the net proceeds in this manner, until a predetermined depth has been reached. In the future the depth of expansion will be determined by a model of the apprentice's capabilities. Further expansion can, of course, occur during execution of the plan if the apprentice requires more detailed instructions.


S
S
Connect Pump To Platform
J
S
Connect Pump Brake To Pump
Connect Pump Brace To Belt Housing Cover
J
S
Connect Pump Pulley To Pump
J
J
S
Connect Belt To Pump Pulley
Connect Belt To Motor Pulley
J
S
Connect Belt Housing Cover To Belt Housing Frame
J

(c)

SA-3805-9b

FIGURE 9 PROCEDURAL NET OF THE DEMONSTRATION SYSTEM (Continued)



(d)

SA-3805-9c

FIGURE 9 PROCEDURAL NET OF THE DEMONSTRATION SYSTEM (Concluded)

b. A Sample Dialog

A sample dialog was given in Section I.B.3 above. It exhibited an interaction between the CBC expert (E) and the apprentice (A) and contained a few annotations and explanations of the interaction. This type of dialog is produced from the execution of a previously-constructed procedural net, although it is possible to have the net constructed "in real time" as the apprentice gives answers to the instructions of the expert. A given procedural net can produce many variations of dialog according to the level of experience of the apprentice. This makes it a powerful base on which to build the rest of the system's capabilities, as will be described in succeeding sections of this report.

C. A Geometric Model Used for Pointing in the CBC Project

1. Introduction

We have devised a method of computer modeling of the shapes of mechanical parts and assemblies, and we have devised a number of algorithms for manipulating the models. These have been incorporated into a demonstration program modeling the air compressor. The program constructs models of the compressor from symbolic descriptions, displays an image in perspective, and may be used to identify or point to various parts of the compressor, by means of a television camera and laser pointer.

The computer representation of complex assemblies is built from primitive solids (cubes, wedges, and cylinders.) In this respect, the models are similar to those described by Braid [24] and others [25, 26]. The principal characteristics which we believe to be unique in this implementation are the use of symbolic "attachment points" to specify the relative positions of parts with respect to one another, and the retaining of symbolic and descriptive information along with purely geometric information.

We use the models in the following ways: An apprentice may "point to" various parts of the compressor by touching that part with a wand with a small light bulb at its tip. The computer responds by naming the part. Alternatively, the apprentice may ask the computer to aim the laser pointer at a named part.

The system at present may be described as "graphics oriented", in the sense that it infers appearances from geometric models, rather than inferring geometry and structure from appearances as a "vision oriented" system would. We are in a preliminary phase in the development of the modeling capability. As we gain experience, the system will be extended to provide a basis for scene understanding in the workshop domain, and to do geometric reasoning for planning purposes.

The demonstration system consists of two modules (or "forks" in TENEX parlance), one operating in a LISP environment and one operating under SAIL. The LISP portion manages symbolic descriptions to determine the structural and spatial relations among parts. The SAIL portion keeps the specific geometric descriptions and absolute spatial positions, and also contains algorithms for manipulating visible outlines. The symbolic/structural data structures and algorithms are described in the next subsection, and the geometric/spatial in the one following.

2. Symbolic and Structural Models

The basic unit of structure in our modeling system is the "part". Descriptions of complex objects are built hierarchically from simpler objects. Thus a part may represent something as simple as a primitive building block, or as complex as the entire workstation. Associated with each part are a number of properties that describe its structure and position, and define attachment points for use in assembling parts into higher-level assemblies.

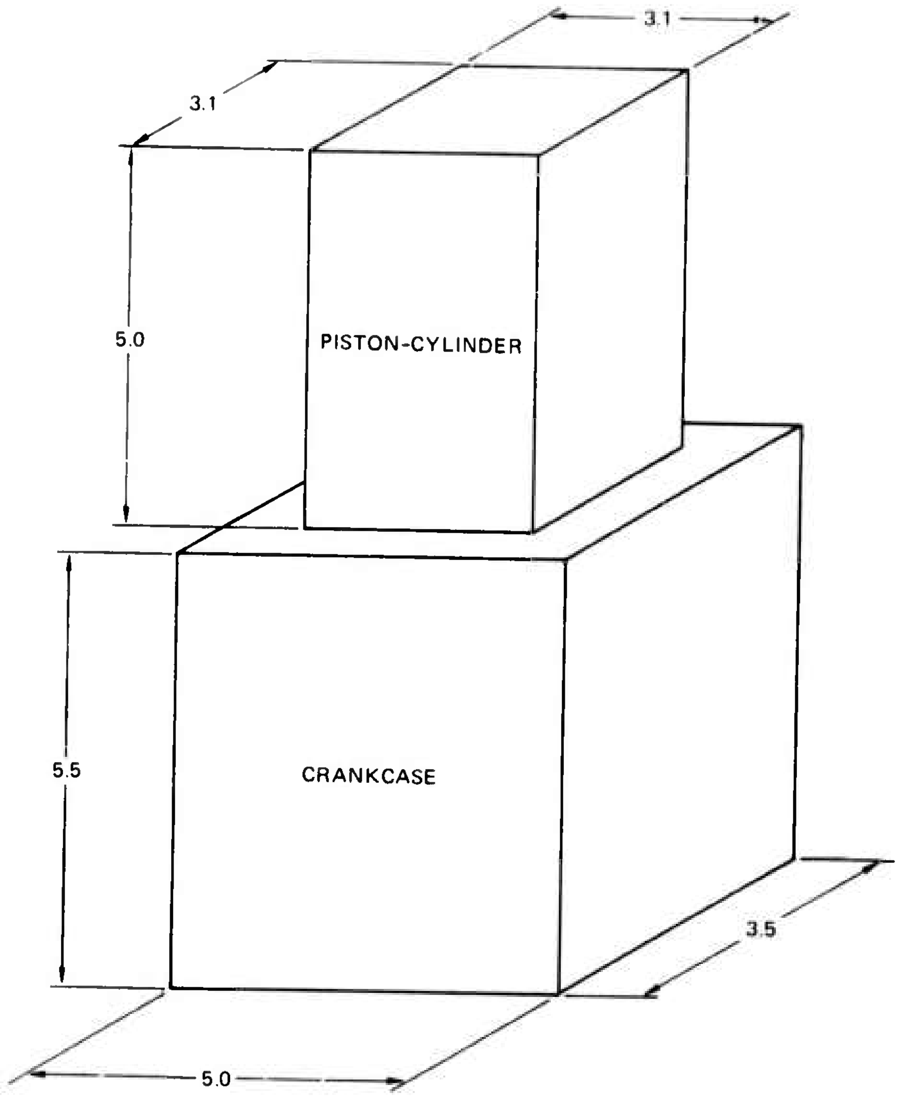
Two kinds of parts are used. "Model parts" make use of dummy parameters to specify their variable dimensions, and are usually described in a canonical position. "Actual parts" are copies of model parts that have a specific position and orientation in space and dummy parameters replaced by actual numbers. Model parts are created by a programmer or designer in specifying a piece of equipment; actual parts are created by the program. In either case, all information about a part is carried on the property list of the LISP atom representing the part.

The system is best described by example. Consider the description of the air compressor's pump. For the purposes of geometric modeling, the pump is modeled as two rectilinear blocks stacked as shown in Figure 10. The model part PUMP contains on its property list the following:

```
    STRUCTURE
         [(CRANKCASE (BRICK 5.0 3.5 5.5))
          (PISTON-CYLINDER (BRICK 3.1 3.1 5.0) (REF CRANKCASE TOP))]
```

This description says that the pump is the union of two simpler parts. The first item in each sublist is a symbolic name to be used within the context of the pump to refer to the subparts. The two subparts of the pump are assigned the symbolic names CRANKCASE and PISTON-CYLINDER. The second item in each sublist describes how to build each subpart. The CRANKCASE is to be modeled as a brick of dimensions 5.0 x 3.5 x 5.5 inches and the PISTON-CYLINDER as a brick of dimensions 3.1 x 3.1 x 5.0 inches.

The remainder of each sublist tells how the subparts are to be positioned within the assembly. Since the crankcase description contains no explicit position, the location of the



SA-3805-10

FIGURE 10 GEOMETRIC MODEL OF PUMP

crankcase is the same as the location of the pump itself. The location information for the PISTON-CYLINDER states that it is to be placed at the top of the crankcase. To find the definition of "top", it is necessary to examine the description of BRICK. The complete description of BRICK is as follows:

```
    STRUCTURE PRIMITIVE
    PARAMETERS (WIDTH DEPTH HEIGHT)
    ATTACHMENTPOINTS
        [(BASE (MTRSP 0 0 (MINUS (FQUOTIENT HEIGHT 2))))
         (TOP (MTRSP 0 0 (FQUOTIENT HEIGHT 2)))
         (BACK (MTRSP 0 (FQUOTIENT DEPTH 2) 0) (MROT "X" - 90.0)))]
```

The attribute PRIMITIVE under the property STRUCTURE implies that there are no subparts to a brick. (There exist routines in both the LISP and SAIL programs for dealing geometrically and spatially with bricks and other primitive parts.) The PARAMETERS list contains symbolic names for the brick's dimensions, which are indicated in Figure 11. Finally, attachment points are provided for placing other objects on or near an example of a brick. Since our description of the pump depended on the TOP of a brick, let us examine its attachment points in greater detail.

An attachment point is a rule for finding a specific point on the surface of a part from an arbitrary point on or within the part. A brick is defined geometrically in the SAIL fork in a

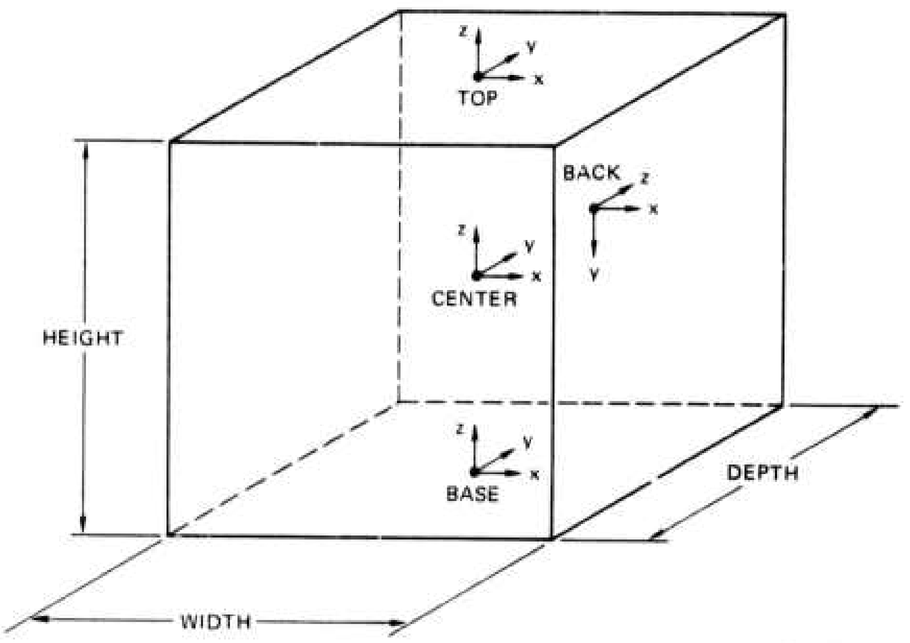


FIGURE 11 BRICK

canonical position and orientation centered on the origin of coordinates. However, it is more convenient to think of the location of a brick as the location of a point on which the brick may be placed. The BASE attachment point allows us to perform this transformation automatically. When we wish to specify that a cube of 2 inches is to be placed on a table 30 inches off the floor, the BASE attachment point tells us the base is located HEIGHT/2 (or 1 inch) below the cube's center. (The function MTRSP represents a translation in x, y and z.) Thus the center is one inch above the tabletop, or 31 inches above the floor. Furthermore, the top of the cube is located one inch above the cube's center, or 32 inches above the floor. (When a part contains no explicit BASE attachment point, as in the case of the pump, the null transformation is assumed. The BASE of the pump is the same as the point on which the CRANKCASE is to be placed.)

The attachment point BACK is interesting in that it is a combination of two primitive transformations. To attach a part to the back of a brick, must be first moved DEPTH/2 back from the brick's center, then rotated 90 degrees about the x-axis so that its base may rest on the vertical back face of the brick. (The function MROT generates a rotation about an axis.)

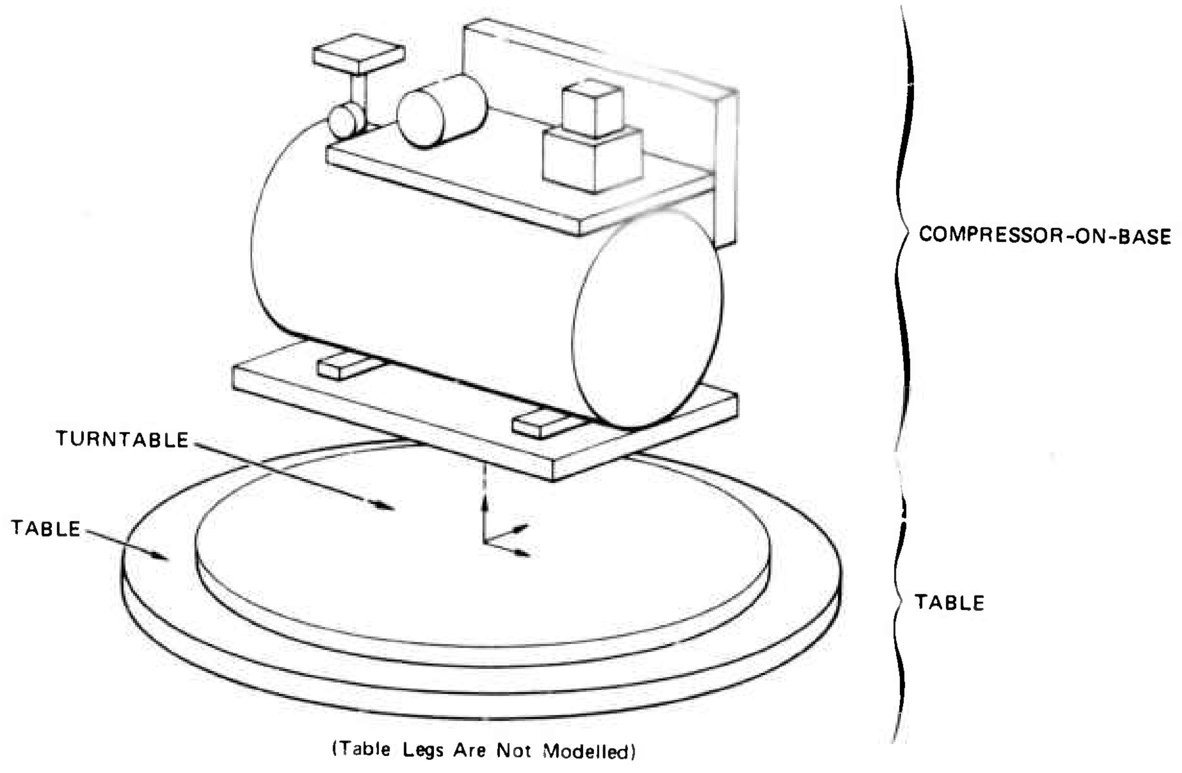
A fair amount of processing is necessary to determine where a part is to be placed. But the main purpose in the use of attachment points is that once they are defined, the programmer need not be aware of it. He need only specify the symbolic name of an attachment point and the computation proceeds automatically.

Parameters have not been well exploited in our description of the air compressor. Nevertheless we expect them to be very useful when we model to a greater level of detail. For example, a single structural description may specify a generic BOLT, with length, diameter, and head type to be supplied in each specific instance. Washers, nuts, bearings, shafts, and other common parts may also be specified in this way.

Using the mechanisms described above, model descriptions of the relevant objects in the workstation may be built up. Such a description is shown in Figures 12(a) through 12(h).

```
ROOM:  STRUCTURE [ (TABLE (TABLE)
                          (MTRSP 6.0 30.0 0))
                   (COMPRESSOR (COMPRESSORONBASE)
                          (REF TABLE TOP)) ]


TABLE:  STRUCTURE [ (TABLE (CYLINDER .7 54.0)
                          (MTRSP 0 0 27.6))
                    (TURNTABLE (CYLINDER .7 36.0)
                          (REF TABLE TOP)
                          (MTRSP 0 0 .95)) ]
        ATTACHMENTPOINTS [ (TOP (REF BASE TOP)) ]
```



(a) ROOM

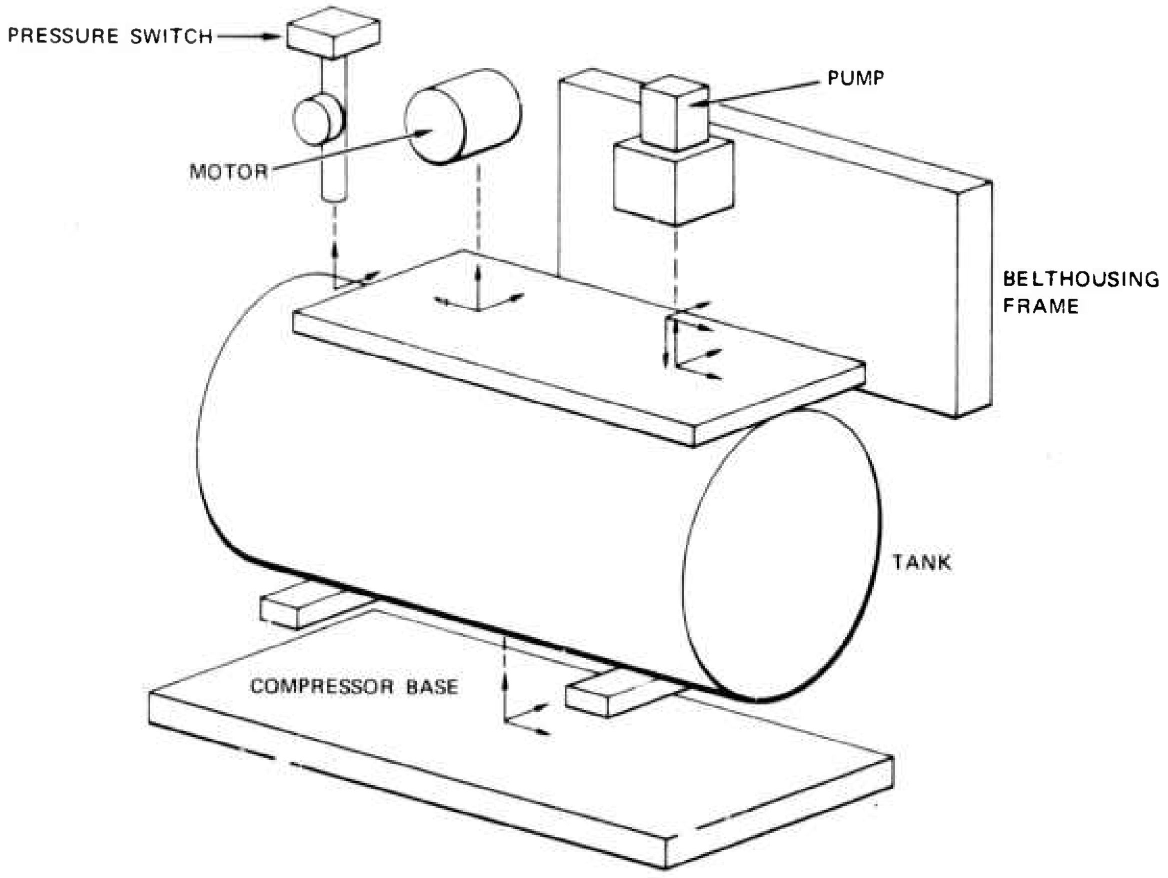
SA-3805-12a

FIGURE 12 GEOMETRIC MODEL OF WORKSTATION

```
COMPRESSORONBASE:
        STRUCTURE [ (COMPRESSORBASE (COMPRESSORBASE))
                    (COMPRESSOR (COMPRESSOR)
                        (REF COMPRESSORBASE TOP)) ]

COMPRESSORBASE:
        STRUCTURE [ (BASE (BRICK 31.625 15.0 1.5)) ]
        ATTACHMENTPOINTS
                  [ (TOP (REF BASE TOP)) ]

COMPRESSOR:
        STRUCTURE [ (TANK (TANK))
                    (PUMP (PUMP)
                        (REF TANK PUMPPLACE))
                    (MOTOR (MOTOR)
                        (REF TANK MOTORPLACE))
                    (BELTHOUSINGFRAME (BELTHOUSINGFRAME)
                        (REF TANK PLATFORMBACK))
                    (PRESSURESWITCH (PRESSURESWITCH)
                        (REF TANK PRESSURESWITCHPLACE)) ]
```

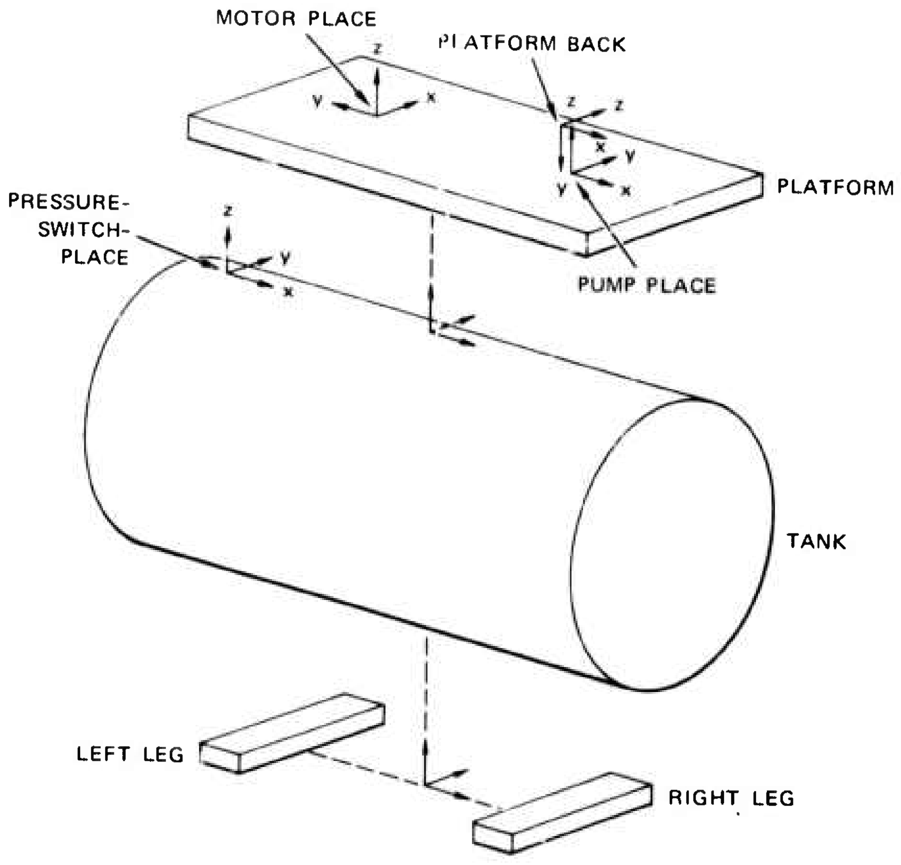


(b) COMPRESSOR

SA-3805-12b

FIGURE 12 GEOMETRIC MODEL OF WORKSTATION (Continued)

```
TANK:  STRUCTURE [ (TANK (HORIZONTALCYLINDER 26.0 12.4)
                        (MTRSP 0 0 1.6))
                   (LEFTLEG (BRICK 2.5 8.0 3.1)
                        (MTRSP -8.0 0 0))
                   (RIGHTLEG (BRICK 2.5 8.0 3.1)
                        (MTRSP 8.0 0 0))
                   (PLATFORM (BRICK 20.5 9.125 3.15)
                        (REF TANK TOP)
                        (MTRSP .375 0 -2.5)) ]
       ATTACHMENTPOINTS
                 [ (PUMPPLACE (REF PLATFORM TOP) (MTRSP 5.85 2.45 0))
                   (MOTORPLACE (REF PLATFORM TOP)
                        (MTRSP -3.15 0 0) (MROT "Z" -90.0))
                   (PLATFORMBACK (REF PLATFORM BACK))
                   (PRESSURESWITCHPLACE (REF TANK TOP)
                        (MTRSP -11.5 0 0)) ]
```

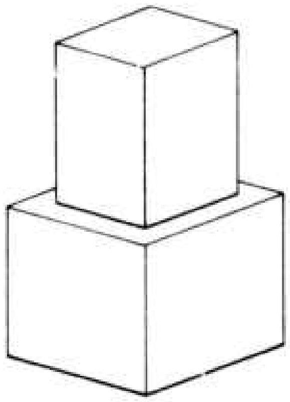


(c) TANK

SA-3805-12c

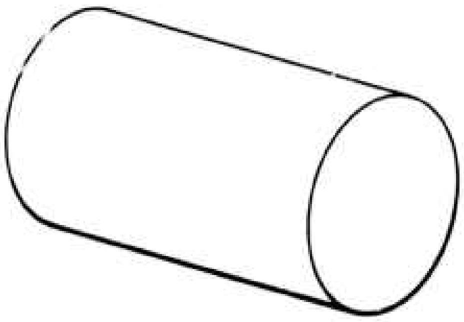
FIGURE 12 GEOMETRIC MODEL OF WORKSTATION (Continued)

```
PUMP:  STRUCTURE [ (CRANKCASE (BRICK 5.0 3.5 5.5))
                   (CYLINDER (BRICK 3.1 3.1 5.0)
                     (REF BOTTOM CYLINDER)) ]
```



(d) PUMP

```
MOTOR:  STRUCTURE [ (MOTOR (HORIZONTALCYLINDER 6.75 5.5)
                           (MTRSP 0 0 .15)) ]
```

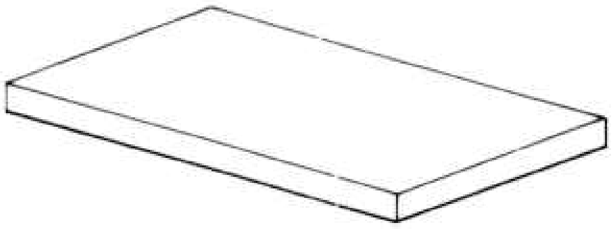


(e) MOTOR

SA-3805-12d

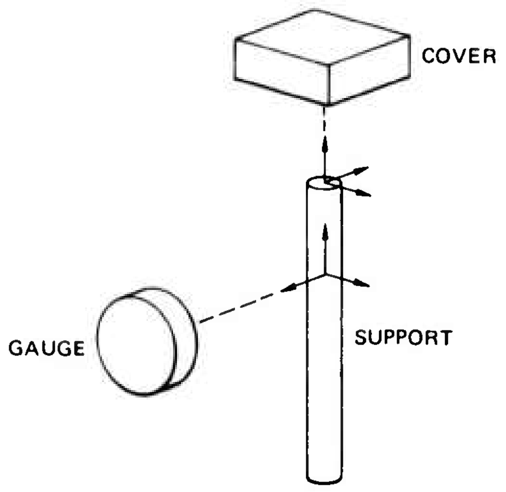
FIGURE 12 GEOMETRIC MODEL OF WORKSTATION (Continued)

```
BELTHOUSINGFRAME:
        STRUCTURE [ (BELTHOUSINGFRAME (BRICK 21.0 3.0 12.0)
                        (MROT "X" 90.0)
                        (MTRSP 1.1 1.5 -.7)) ]
```



(f) BELT HOUSING FRAME

```
PRESSURESWITCH:
        STRUCTURE [ (SUPPORT (CYLINDER 3.7 1.0))
                    (COVER (BRICK 3.6 2.7 3.0)
                        (REF SUPPORT TOP))
                    (GAUGE (CYLINDER 1.0 2.2)
                        (MTRSP 0 -.5 2.4)
                        (MROT "X" -90.0)) ]
```

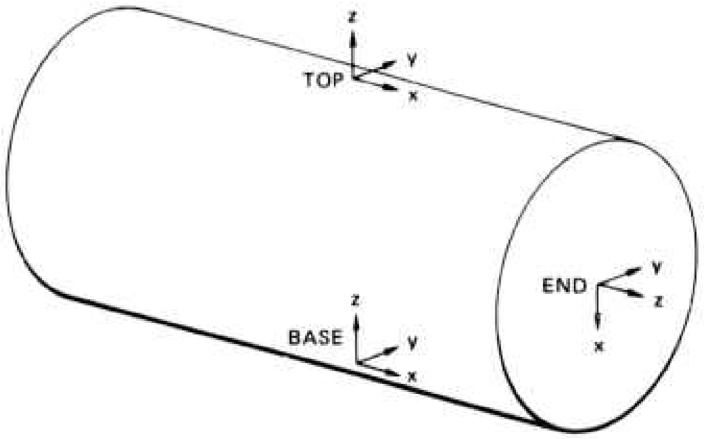


(g) PRESSURE SWITCH

SA-3805-12e

FIGURE 12 GEOMETRIC MODEL OF WORKSTATION (Continued)

```
HORIZONTALCYLINDER:
        PARAMETERS (LENGTH DIAM)
        STRUCTURE [ (CYLINDER (CYLINDER LENGTH DIAM)
                        (MTRSP (MINUS (FQUOTIENT LENGTH 2))
                                0
                                (FQUOTIENT DIAM 2.0))
                        (MROT "Y" -90.0)) ]
        ATTACHMENTPOINTS
                    [ (BASE NIL)
                     (TOP (MTRSP 0 0 DIAM))
                     (END (REF CYLINDER TOP)) ]
```



(h) HORIZONTAL CYLINDER

SA-3805-12f

FIGURE 12 GEOMETRIC MODEL OF WORKSTATION (Concluded)

Since descriptions exist only as property lists attached to the atoms representing model parts, and since any part may be represented several times in a given scene or assembly with different actual parameters, a function is necessary that will create actual parts as copies of the model parts. The function CREATE accomplishes this in two passes through the data structure.

CREATE must be given a part name from which to create a copy, actual parameters (if appropriate), and a position in which to place the completed assembly. In the first pass, the hierarchy of the part is explored, copies of each part created, actual parameters evaluated, and relative positions calculated within each level of hierarchy. In the second pass, the relative positions are accumulated to give an acutal position for each member of the assembly. The primitive objects that form the terminal nodes of the hierarchy are passed to the SAIL segment with their actual dimensions and positions for incorporation into the polyhedral data structure. The result of the call on CREATE is a copy of the model part, with all actual parameters, sizes, and positions evaluated.

What we have, then, is a method of specifying a hierarchical organization of parts and subparts, using some very simple primitives as the terminals. This method is more human oriented than previously published methods in that it allows the use of symbolic attachment points, and it allows the use of parameters to specify dimensions and other variable information that will be supplied at evaluation time.

The data structures and functions already implemented could form the basis for an interactive parts design system. Although we have not done so, it would be relatively easy to add facilities to display a continuously updated picture of an assembly as the programmer or designer specifies a new part. In fact, we intend to implement a minimal such facility to specify new parts when a new domain for the CPC project is chosen.

A class of parts that have not been well modeled by our method are nonrigid parts, such as cables and belts. To deal with these in the future, we propose to implement a new class of primitive called "snakes." These will consist of a cross section description and a set of constraints on a space curve along which the cross section is traced. Such an approach to object descriptions has been described by Agin [27]. Although details have yet to be worked out, it seems likely that specification of the length of a snake and the positions of its end points will be useful for most of the purposes we envision.

3. Geometric and Spatial Models--Visible Outlines

This section discusses the algorithms that use the symbolic models in order to identify and point to parts of a real compressor with the TV camera and laser pointer. But first, a brief discussion of the SAIL-based data structure is in order.

Information about polyhedra is contained in short contiguous blocks of core that we call nodes. A node is 14 words long, and contains a mix of two types of data: addresses of related nodes, and floating point numbers representing dimensions, coordinates, or vectors.

Corresponding to each actual primitive part in the LISP data structure is a polyhedral representation called a "body node." (Cylinders are approximated by eight-sided prisms. Bricks and wedges are themselves polyhedra.) The bodies are linked together hierarchically to form "objects," duplicating some of the structural information of the LISP parts. Below the level of bodies are "faces," "edges," and "vertices."

The structure of a body, and its faces, edges, and vertices follows the winged-edge representation described by Baumgart [28], which may be summarized briefly as follows.

Most of the topological connectivity information about the body is carried in edge nodes. Each edge node contains pointers to the faces on each side of the edge, to the vertices on each end, and to four neighboring edges. The vertex nodes contain the actual x-y-z coordinates of the vertex, while the face nodes contain a 4-element vector giving the plane equation of the face in homogeneous coordinates. In addition, there is space in each face and vertex node for transformed coordinates giving the same information in the coordinate frame of the TV camera or laser pointer. (Frequent mention will be made in this section of homogeneous coordinates, and of translations, rotations, and projective transformations based on homogeneous coordinates. It is beyond the scope of this report to describe them. The interested reader is advised to refer to Duda and Hart [29].

Associated with each object is an "enclosing sphere". Contained in the object node are the radius and the location of the center of a sphere within which the object will be entirely enclosed. These spheres are extremely useful in some of the algorithms to be described later for minimizing search. For an object composed of only one primitive polyhedron, this sphere may be calculated from the dimensions and location of the polyhedron. In the process of combining two or more objects to form a more complex one, a simple geometric calculation will find a sphere (but not necessarily a minimal one) that encloses the union of the enclosing spheres of the individual parts.

A straightforward use of the polyhedral models is in the presentation of a visual display of the compressor. A set of subroutines can convert the polyhedral representation of bodies to a set of vectors in homogeneous coordinates. The term "wire model" is appropriate here because, as shown in Figure 13, hidden lines and

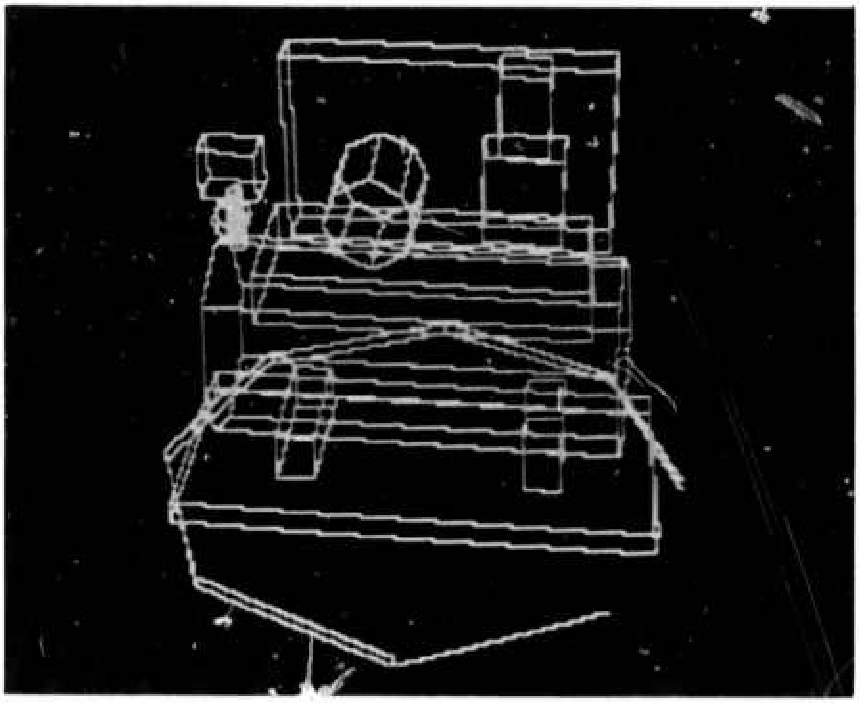

SA-3805-47

FIGURE 13 WIRE MODEL OF THE COMPRESSOR

back faces are not removed, and the display looks like a collection of "wires" along the edges of the model. The user has control (from the keyboard) of the virtual camera through which the scene is viewed. Thus the display may be rotated, translated, shrunk, or enlarged. A special split-screen viewer allows viewing the image in binocular stereo.

The commands to alter the display are not now in a form such that an apprentice could easily control it. But it should not be difficult in the future to design a control language to use such commands as "move left," "zoom in," "tilt up," or "rotate."

The algorithms for pointing at and identifying parts of the compressor depend upon precise knowledge of their positions and orientations. This is because the locations of the polyhedral models are assumed to be correct, and reference is not made to the image of the compressor as such. We presume we can calculate from the LISP model exactly where the image of each part would appear on a TV picture, and make no checks to ascertain that this is actually the case. (We are currently engaged in efforts to identify the compressor in images obtained from the TV camera or from the laser used as a range finder. This information will be used to calculate the actual position of the compressor, and update our spatial models.)

"Calibration" is the procedure by which we calculate the transforms and internal parameters for the laser pointer and TV camera. These transforms are necessary if we are to be able to calculate image coordinates from positions in space and vice versa. A semiautomatic procedure for calibration entails aiming the laser at various places about the room where pieces of graph paper have been precisely located on the wall or table top, and typing into the computer the graph coordinates at which the beam hits the paper. Based on these locations and the deflections applied to the laser, it is possible to calculate (by an iterative "hill climbing" algorithm) a transform that gives the best correspondence between predicted and actual graph coordinates. The TV camera may be similarly calibrated by using the raster coordinates of the bright spot in the image where the laser strikes the graph paper.

To check the correspondence of the model, the actual compressor, and the TV calibration, it is sometimes useful to superimpose on the same display a digitized TV image and the "wire model" of the compressor. Figure 14 shows such a display.

Once correspondence is established, we can find the intersection of a ray in space with the compressor. This is how we identify parts pointed to by the apprentice. He points with a wand with a small light bulb at its tip. The light bulb is identified as the brightest spot in a TV picture of the scene. (The laser pointer might also create a bright spot in the picture, depending on where it is aimed. But a green filter in front of the camera eliminates this

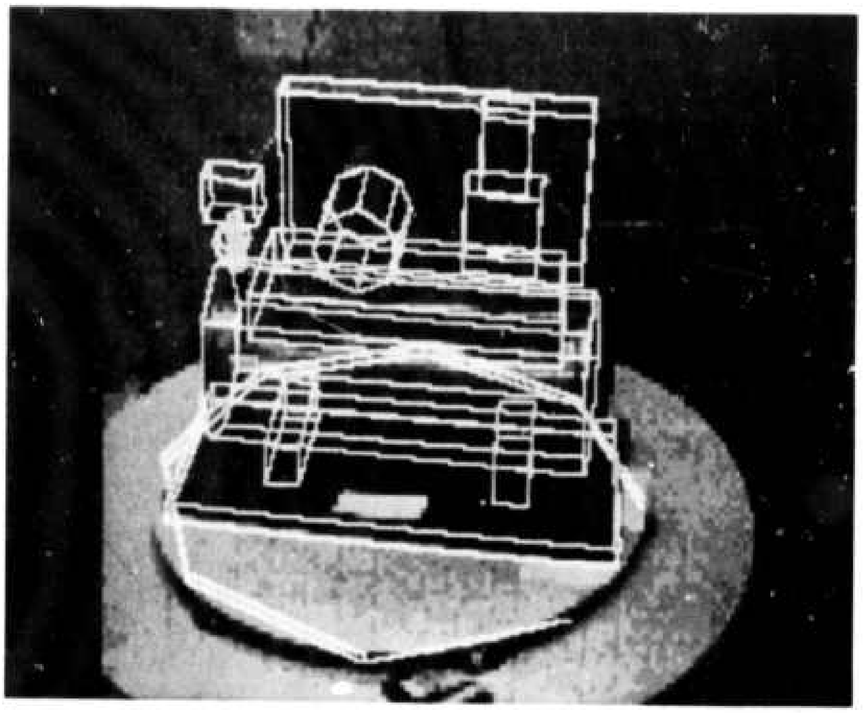

SA-3805-48

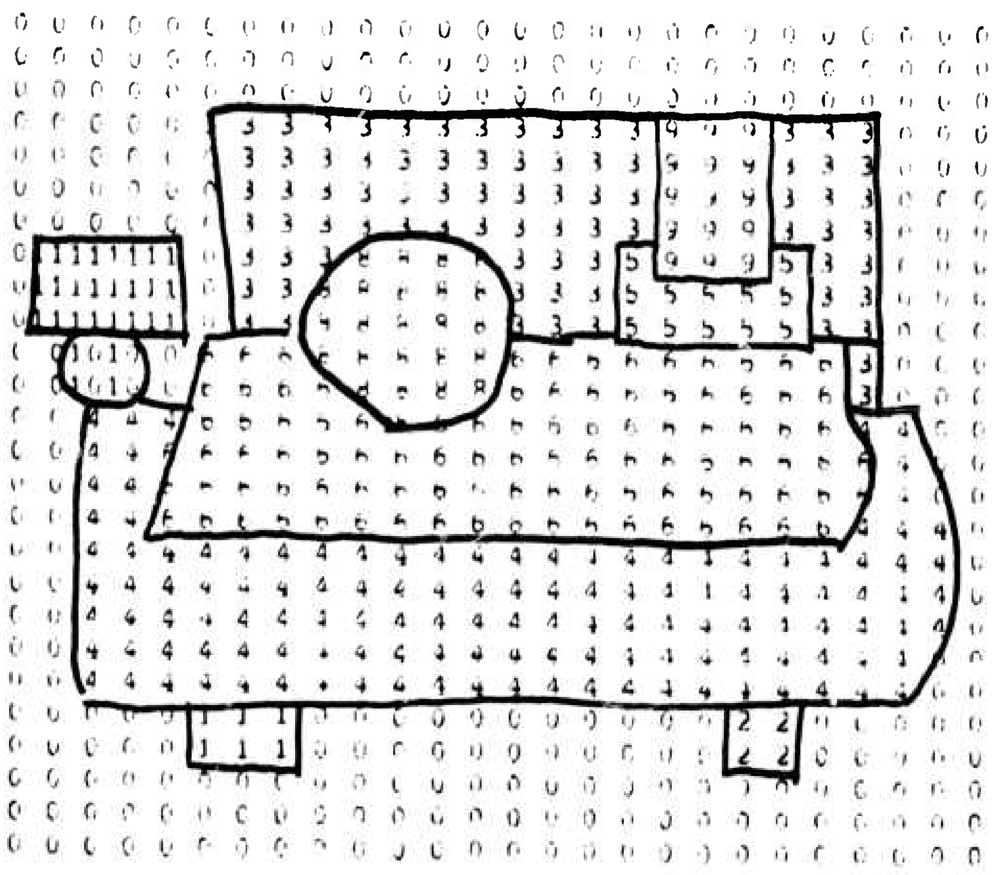
FIGURE 14 WIRE MODEL SUPERIMPOSED ON TV IMAGE

possible ambiguity.) The straight line connecting the camera center and the light bulb defines the ray in space, whose intersection with the compressor model we are to find.

The light bulb must be touching or in close proximity to the part the apprentice wishes identified. (To make it possible for him to point to the part from a distance would require determining the position of the wand in three dimensions. Locating just the tip of the wand entails only a two-dimensional determination of its image coordinates.)

For intersecting the ray with the model, it is useful to work in the projective coordinate system of the TV camera. The ray, in this coordinate space, is a line of constant I and J (the image coordinates measured for the bright spot.) Starting with the object node representing the entire workshop domain, objects whose transformed enclosing spheres do not include the coordinates of the ray are eliminated, and those that do are broken down into subobjects, until we arrive at a set of candidate bodies that cannot be ruled out on the basis of their enclosing spheres. Further analysis of the polyhderal representation of each body yields either (1) the distance from the camera at which the ray strikes a face of the polyhedron or (2) the fact that the ray does not strike the body at all. Based on these distances, the body closest to the camera is identified as the one pointed to.

An alternate procedure is sometimes used. This entails subdividing the scene (as seen by the TV camera) into a coarse "visibility matrix" of 40 x 40 points. Agin and Nitzan [30] describe a procedure by which a matrix representation of the edges of a polyhedron may be derived, and the interior of the resulting outline marked or "colored in" . All of the objects in the scene are sorted by their distance from the camera. Then starting with the farthest object, the polyhedra that make up the object are "colored" with a number identifying that object. Where the images of two objects overlap, the number of the nearer overwrites the number of the farther. Figure 15 shows the result of applying this procedure to the model of the compressor.



```
 1 = TANK LEFTWING
 2 = TANK RIGHTWING
 3 = BELLHOUSINGFRAME
 4 = TANK CYLINDER
 5 = PUMP BOTTOM
 6 = TANK PLATFORM
 7 = PRESSURESWITCH SUPPORT (not visible)
 8 = PUMP CYLINDER
 9 = PUMP TOP
10 = PRESSURESWITCH GAUGE
11 = PRESSURESWITCH COVER
```

SA-3805-43

FIGURE 15 VISIBILITY MATRIX

With this precomputed matrix, determining what is being pointed to by the light wand requires only converting the coordinates of the bright spot into the appropriate indices of the matrix, and retrieving the identifying number stored there.

This visibility matrix procedure provides a faster response to each identification request, at the cost of increased computation during the model-building phase. In addition, the matrix must be recomputed each time the relative positions of the camera and compressor change. The sorting of objects by distance of sphere centers from the camera is not always guaranteed to work, but for the cases we have tried, the results have been adequate. Thus, although the visibility matrix approach is the one we generally use, we are aware of its limitations, and have the more reliable ray intersection method to fall back on should cases arise where the faster method fails.

Regardless of which procedure is used, the SAIL fork passes to the LISP fork the address of the object node it found. By referring to a table of node-part correspondences, a particular primitive part is retrieved. There remains the problem of translating this primitive into a meaningful response to the original question, "What is this?" A possible solution would be to return the entire ancestry of the part in its hierarchy, such as (ROOM COMPRESSOR COMPRESSOR-ON-BASE COMPRESSOR PUMP PUMP PISTON-CYLINDER BRICK). But such an answer is unnecessarily long and redundant. For lack of a better model of what is wanted, our solution is to reply with the smallest nonprimitive part to which the primitive belongs, and the symbolic substructure name of the primitive. The above example reduces to (PUMP PISTON-CYLINDER). A few special-case rules take care of instances where such an answer would be inappropriate or ridiculous. Clearly, here is a case where models of the dialog and a model of the apprentice would help in formulating an appropriate response.

The converse of part identification is pointing. An apprentice may ask "where is the platform?" and expect the computer to point to the platform (on top of the tank) with the laser pointer. Satisfying this request entails two steps: finding an outline of the platform (taking into account any parts that may be in front of the platform and hide a portion of it), and locating a point near the centroid of the outline at which to point.

The first part may be accomplished by using a visibility matrix such as the one previously shown in Figure 15. (In this case, the matrix must be computed in laser coordinates rather than TV coordinates, so a separate matrix is necessary.) Or the outline of the desired part may be obtained by the method of Ref. 31. Those objects whose enclosing spheres indicate the possibility of hiding a portion of the platform will be similarly outlined and their "colored in" portions subtracted from the original. The procedures for recognizing occlusion and for subtracting the marked areas are described in detail by Nitzan [31]. Given the matrix representation of the visible portion of the desired object, its centroid may be found by the method of "chamfering," described in Ref. 32. The x and y indices of the centroid point are converted to the appropriate laser deflection constants, and the laser pointer will then move so as to point near the center of the part.

D. Deductive Retrieval Mechanisms for State Description Models

1. Introduction

This section describes some modeling facilities that have been developed as part of the CBC. The models that we are concerned about here could be descriptions of any environment of interest at specific instances in time. Each such description is said to model a "state" of the environment, and a state can be transformed into a new state by an event that alters the environment. For the CBC, these models describe the state of the workstation--including the device, tools, test equipment, and so on--and the events are primarily maintenance and repair operations performed by the apprentice.

Programming facilities for querying state description models and for updating them to reflect the occurrence of an event are vital in many AI systems, particularly those that do question answering and those that do automatic generation and execution monitoring of plans. Planners, for example, use these models to simulate potential operator (event) sequences and to investigate their consequences.

The modeling mechanisms to be described here are basically extensions and modifications of facilities typically found in recent planning programs implemented in languages such as CONNIVER [33], PLANNER [34], and QA4 [35]. The need for such additional knowledge representation mechanisms is evident as AI projects continue to move in the direction of considering more complex task domains. The presentations here are meant to add an increment to our ability to design and build such large systems.

2. Expressions, States, and Contexts

Our modeling system is implemented as an extension of the QLISP programming language [18]. Each state description model can be thought of as a set of QLISP expressions, with each expression having a truth value associated with it. An expression typically is a statement of a relationship among entities in the task domain such as objects, concepts, or other relationships. For example, the expression (CONNECTED PUMP PLATFORM) is a statement that the pump is connected to the platform, and the expression (FASTENER (CONNECTION PUMP PLATFORM) BOLT:1) is a statement that BOLT:1 is a fastener of the pump-platform connection.

We use the QLISP context mechanism, which allows the system to build and manipulate a tree of state descriptions without having to create and maintain a complete copy of each state. Each node in this "context tree" denotes a state. To represent the new state that is produced by the occurrence of an event in some state S1, the system creates a new node in the context tree as a direct descendent of the S1 node. All information in state S1 that is not explicitly changed in the new state is assumed to also hold in the new state. That is, each state inherits model information from the state that is its parent in the context tree.

The QLISP context mechanism depends on the fact that all information associated with an expression in the data base is stored on the expression's property list. The presence of an expression in the data base implies only that there has been a reference to the expression (as with LISP atoms), but says nothing about any of its properties, including its truth value. A QLISP expression actually has a collection of property lists, one for each state in which some information has been stored about the expression. Hence, when an expression is asserted as being true in some state S1, TRUE is stored as the value of the property TRUTHVALUE on the expression's S1 property list.

3. Querying State Description Models

a. Truth Values

A state description model is a source of information about a particular situation, and its primary use is as a data base for answering queries about the situation. Our modeling system interacts with its users (both people and programs) as if QLISP expressions with truth values attached were the only representations being used. Hence, all queries from outside the modeling system concern the truth value of expressions in some given state.

When answering a query about a particular expression in some given state, the system searches for a truth value. The search begins with the property list for the given state. If the property TRUTHVALUE has no value on that property list, the property list for the given state's parent (in the context tree) is checked. The search continues in this manner until a value is found or until all the states in the context are considered. If no value is found, the search returns UNKNOWN as its result. Since any expression can be stored as the value of property TRUTHVALUE, this retrieval mechanism allows use of an N-valued logic. For examrle, one could have "fuzzy" truth values represented as integers from -100 to +100. For our models, we currently are using a 3-valued logic that allows the system to distinguish expressions that are "known true", "known false", or "have unknown truth value" in any given state. This is the simplist logic that meets a modeling system's needs since state description models are inherently incomplete, and it is important for the system to be aware of what it does not know as well as what it does know.

b. Generators Instead of Backtracking

QLISP provides facilities for associatively retrieving expressions from the data base that match any given pattern, where a pattern is defined to be an expression that contains unbound variables. The QLISP statements for querying the data base use this pattern matching facility and are similar to the query statements found in PLANNER and QA4. They are designed to find a single instance of a given pattern. To cause the pattern matcher to continue its search and obtain another such instance, the user's program must return to the query statement via the language's backtracking mechanism (i.e., by "failing").

Using backtracking in this way to sequence through a class of expressions that all match a given pattern has severe limitations in that it ties the sequential production of each expression to the control structure of the user's program. In particular, it requires that the same portion of the user's program be executed for each expression (namely, the statements immediately following the query statement). Also, since all the backtrackable effects of that portion are being "undone" after each failure, it makes cumbersome the saving of results for each expression generated. Such a backtracking mechanism is best suited to a "generate-and-test" situation where the user desires a single expression that not only passes the query statement's tests, but also passes additional tests included in the user's program.

We have adopted the CONNIVER solution to these limitations in our modeling system by providing functions that are generators of expressions from the data base. For example, there is a generator version of the QLISP IS statement called GEN:IS that finds instances of a given pattern having truth value TRUE in a given state. Each time a generator function such as GEN:IS is called, it produces as many expressions as is convenient for it. These expressions are put on a "possibilities list" along with a "tag" that indicates how the generator can be restarted when more expressions are requested, and this possibilities list is returned by the generator as its value.

If the function TRY:NEXT is called with a possibilities list as an argument, it will remove the first expression from the list and return it as a value. If the possibilities list contains no expressions, then TRY:NEXT attempts to produce new ones by using the tag to restart the generator. Since each call to TRY:NEXT can be made from anywhere in the user's program, generators of this form successfully separate the production of a next data element from the processing that is done on each element.

Consider, for example, a set of queries concerning which components are connected to the pump in state S1. They can be initialized as follows:

```
(SETQ PL (GEN:IS (CONNECTED PUMP ←C) S1)).
```

Then whenever one of these components is needed, evaluation of (TRY:NEXT PL) will return a true instance of the pattern (CONNECTED PUMP ←C) and will set the value of the QLISP variable C to be the "found" component.

We have implemented programming facilities to support the writing and use of generator functions using INTERLISP FUNARGS. A FUNARG is a data object that conceptually represents a copy of a function and a private data environment for that copy. This FUNARG implementation allows the definition of a generator function to include a set of variables (i.e., a data environment) whose values will be saved and restored each time the generator is restarted. These "own variables" allow the generator function to save pointers indicating where it is in its search for generatable items. The FUNARG is added to the possibilities list as the "tag" that TRY:NEXT uses to restart the generator. Included in the implementation are CONNIVER-style functions such as NOTE, AU-REVOIR, ADIEU, and TRY:NEXT, which make the definition and use of generators convenient and practical.

c. The Query Functions

We can now describe our model querying mechanism. The following query functions are available:

```
(DEDUCE:ONE <pattern> <context>),
(DEDUCE:EACH <pattern> <context>),
(DEDUCE:ALL <pattern> <context>),

(REFUTE:ONE <pattern> <context>),
(REFUTE:EACH <pattern> <context>),
(REFUTE:ALL <pattern> <context>).
```

The DEDUCE functions find instances of the pattern that are true in the given context, and the REFUTE functions find instances of the pattern that are false in the given context. The :ONE functions find only a single instance and are not restartable; the :EACH functions are generators and return possibilities lists; and the :ALL functions return a list of all the findable instances.

Known truth values are usually not all explicitly stored in a model. Instead, the user provides derivation functions that compute them when they are needed. These functions may embody formal theorem proving strategies or simply be statements of implicational rules derived from the semantics of the task domain.

They serve to extend each model in the sense that, from the calling program's point of view, the derived instances of a pattern are indistinguishable from instances actually found in the model.

Our query functions are similar to a PLANNER or QA4 GOAL statement in that they first use the pattern matcher to find suitable instances of the pattern in the data base and then, if more instances are needed, they call user supplied functions to attempt derivations of the desired instances. These functions are assumed to be generators that produce derived instances of the pattern.

A typical deduction function in the CBC system finds and generates true instances of patterns of the form (POSITIONED ←X ←Y) by using DEDUCE:EACH to find true instances of the pattern (ATTACHED $X $Y), since components that are ATTACHED are assumed to be POSITIONED. A typical refutation function finds and generates false instances of patterns of the form (POSITIONED ←X ←Y) by using DEDUCE:EACH to find true instances of the pattern (REMOVED $X $Y), since components that are REMOVED are assumed to be not POSITIONED.

These derivation functions are the user's primary means of expressing the semantic links among the relations occurring in the state description models. Also, they can provide an interface to information that is stored in representations other than

QLISP expressions. That is, it may be much more convenient and efficient to store some information in arrays, trees, or on disk files; deduce and refute actions serve as the access functions to these alternate data bases.

d. Storage and Retrieval of Action Functions

The first element of each nonatomic expression in the model is assumed to be the name of a relation (or a QLISP variable that is to be bound to a relation). Therefore, the DEDUCE and REFUTE functions can use relation names as an index to determine which derivation functions should be called. Accordingly, we associate with each relation name two lists of derivation functions that can derive instances of patterns that begin with the relation. One list contains the "deduce actions" used by the three forms of DEDUCE and the other contains the "refute actions" used by the three forms of REFUTE.

4. Saving Derived Results

When a model query causes derivations to be attempted, we want the results of those derivations to be stored and retained in succeeding states as long as they remain valid. In this way the system achieves the maximum benefit from derivations and minimizes unnecessary rederivations.

A model query is an attempt to find true (or false) instances of a given pattern. Each time such an instance is determined, our DEDUCE and REFUTE query functions save the derived result by assigning a truth value to the instance (i.e., put it as the value of TRUTHVALUE on the expression's property list) so that the value will not have to be rederived if it is needed again. For example, if a deduce action for ASSEMBLED determines that the pump is assembled by querying the model about each of the pump's components, then the expression (ASSEMBLED PUMP) will be assigned a TRUTHVALUE of true.

If a query is one of the :ALL forms, or if it is an :EACH form and the generation continues until all derivable instances of the pattern are produced, or if the query pattern contains no unbound variables (and therefore has only one possible instance), then the system also records the fact that all instances of the pattern have been derived. If the same query is repeated, the system will know that the action functions cannot find any new instances and can therefore prevent ill-fated attempts at rederivation. For example, if during a query all the components that are positioned with respect to the pump have been found as instances of the pattern (POSITIONED PUMP ←C), then, when that information is requested in a later query, derivation functions such as the one that looks for components attached to the pump will not be recalled.

These "set completeness indicators" are also frequently useful to indicate the case where there are no derivable instances of a pattern. For example, if all derivation attempts are unsuccessful at determining whether the pump is assembled, then the set of derived instances is empty and marked as complete.

Our algorithms for maintaining these derived results in succeeding states depend on availability of the "support" for each derivation. The "support" for a derived instance is defined to be those expressions from the model that are used as axioms in constructing the derivation. For example, if an action function queries the model for the locations of two objects and concludes that one of the objects is above the other, then the locations of the two objects form the support set for the result. Actually, since any model query may return a derived result, the support set for the "above" result would be the union of the support sets for the two location expres ions.

A derived result remains valid in succeeding states as long as its support remains valid. We therefore have the system do the required maintenance on derived instances in new states by including the following facility:

> Whenever an expression with a known truth value has its truth value changed during a model update, the truth value of each of the expression that it supports is set to UNKNOWN in the new state.

The truth values of these "supportees" may not in fact have changed in the new state, but the derivations that made the truth values known are no longer valid. If a model query needs to know one of the deleted truth values in the new state, a new derivation must be attempted. For example, if (ATTACHED PUMP PLATFORM) is the support for (POSITIONED PUMP PLATFORM) and a detach pump from platform action causes a new state to be created, then the truth value of (POSITIONED PUMP PLATFORM) will be set to "unknown"; the pump may still be in position on the platform, but the justification for the earlier conclusion about the pump's position is no longer valid.

A similar mechanism is used to save and maintain the "completeness indicators" that are attached to derived sets of instances.

5. State Transitions

a. Updating Functions

The models of state changing operators that a system works with must contain sufficient information about the effects of each operator so that they can be simulated and a description produced of the expected resulting state. As in most planning systems, we are assuming that the application of an operator in some state S0 is modeled by producing a new state S1 that is conceptually an updated copy of S0 (i.e., S1 is a direct descendent of S0 in the QLISP context tree). The effects of the operator are indicated by asserting, denying, and deleting expressions in the new state S1.

In our modeling system we provide the following set of model updating statements:

```
(SIM:ASSERT <expression> <context>),
(SIM:DENY <expression> <context>),
(SIM:DELETE <pattern> <context>).
```

SIM:ASSERT (SIM:DENY) changes the truth value of the given expression to TRUE (FALSE) in the state indicated by the given context. SIM:DELETE changes the truth value of all expressions that match the given pattern to UNKNOWN in the state indicated by the given context. These statements also call a set of user supplied functions (like PLANNER antecedent theorems) that typically make additional changes in the new model that are direct results of the assertion, denial, or deletion being done. These user supplied functions play an important role in simplifying operator models in that they allow the user to express in one place side effects of particular assertions, denials, or deletions that always occur no matter what operator does them. In this way, these side effects do not have to be repeated in each operator that causes them to occur.

As in the case of the DEDUCE and REFUTE

functions (and for similar reasons), we have elected to store the user supplied updating functions on each relation's property list. Hence, a relation can have a list of ASSERT:ACTIONS, DENY:ACTIONS, and DELETE:ACTIONS. These lists indicate how a model updating operation should proceed for an expression having the given relation as its first element.

The updating functions also perform the maintenance operations on derived results. This means that if the expression had a known truth value and that value is being changed, then all the expression's supportees must be deleted. Also, checks are made to determine if any set completeness indicators should be removed.

b. Model Updating Using Deduce and Refute Actions

Consider now a consistency checking procedure that could be applied as a standard part of model updating. The purpose of the procedure would be to perform additional assertions, denials, and deletions that are implied by the given expression's truth value change without requiring the user to write additional action functions. For example, if the user has written a deduce action that embodies the rule "X implies Y", then we do not want him to also have to write a deny action for Y that removes from the model any truth values that could be used to derive X. The information necessary to do these changes at the appropriate time is included in the original deduce action.

This procedure would work as follows. The truth value of the expression being updated would be set to UNKNOWN and an attempt would be made to deduce the expression. If this attempt produces a successful derivation, then the new state contains support for the truth of the expression even though it is being denied. The inconsistency can be eliminated by removing the support for the derivation. If the support set has exactly one expression in it, then that expression can have its truth value reversed. (This is the "X implies Y" case where the denial of Y is implying the denial of X.) The reversed truth value would be stored as a derived result with the original expression being updated (Y) as its support. When the support set contains more than one expression, we know that at least one of the expressions must have its truth value changed, but we do not know which one(s). Therefore, the best we can do is to delete (i.e., make truth value unknown for) all of the support expressions.

If the system knows which relations are changeable by events and which ones are true in all states, then it can decrease unnecessary deletions by removing the unchangeable expressions from the support set before considering deletions or a truth value reversal. After the support for the derivation has been removed, a new derivation is attempted and the process is repeated until no new derivation can be found.

This updating procedure does not guarantee consistency in the new state nor does it prevent later changes to the state from introducing new inconsistencies. However, it does automatically take care of many model updating details and it removes all existing inconsistencies in the state that are discoverable by the system's deductive machinery. If the system cannot derive the facts from which an inconsistency follows, then the inconsistency is irrelevant and can safely be ignored.

Obviously, there are situations in which this procedure initiates computationally expensive derivations and causes many unnecessary deletions. Hence, it must be selectively applied. We currently have the entire process under user control by allowing individual specification of which deduce and refute actions are employed to determine truth value deletions and reversals during model updating.

Note that using an "X implies Y" deduce action as a deny action for Y is not the same as writing a deny action for Y that simply denies X. The difference is that in the latter case X would be denied each time Y is denied, and the deny actions associated with X would then trigger off other assertions, denials, and deletions. Such a process could clutter up the model with many irrelevant implications of the denial of Y. However, in the former case where the deduce action is used as the deny action for Y, no changes are made in the model if X cannot be derived; and if X can be derived, only the supporters of the derivation are changed. This means that only those truth values that are actually inconsistant with the denial of Y are changed; no irrelevant implications are stored.

6. The Relations 'AND', 'OR', and 'NOT'

AND, OR, and NOT are "built into" our modeling system in the sense that deduce, refute, assert, deny, and delete actions have been written for each of them. Whenever possible, conjunctions, disjunctions, and negations are decomposed into more primitive forms by the action functions. For example, the assert action for AND also asserts each of the conjuncts, and the deduce action for NOT strips off the NOT from the query pattern and attempts to refute the remaining pattern.

The refute actions for AND and OR translate the query into a call on DEDUCE:EACH by using the rules:

((NOT X1) AND ... AND (NOT Xn)) implies (NOT (X1 OR ... OR Xn), and
((NOT X1) OR ... OR (NOT Xn)) implies (NOT (X1 AND ... AND Xn) .

The deduce actions for AND and OR have a important role to play in that they are the overlords of the derivations of each conjunct or disjunct. They could each be expanded into full problem solving executives that would make use of co-routine facilities to explore alternative derivations in parallel and semantic information to determine the order in which conjuncts or disjuncts are considered. We have experimented with only unsophisticated versions of these actions, but the important point to note is that the query mechanism gives those actions control over the derivation so that the option is there to expand them when needed.

7. Summary

We have described a set of programming facilities for building, maintaining, and querying state description models. These facilities are useful in systems such as planners, question answerers, and simulators. They allow the storage and retrieval of statements with true, false, and unknown truth values, and provide a programming environment that allows derivation rules embodying the semantics of a task domain to be easily added as functions to the system. These rules can also be used to assist in modeling the effects of an operator that creates a new state. Facilities are provided to save the results of these derivation functions, and to delete the results in new states where the derivations are no longer valid. Finally, the semantics of conjunctions, disjunctions, and negations are provided as a part of the system.

E. Use of Voice Input and Output

1. Implementation of Voice Input Using the VIP-100

The CBC System employs a VIP-100 voice recognition device which is capable of recognizing 64 utterances of duration up to two seconds each, when trained for a specific voice. The process is as follows.

First the message data is input. This consists of up to 64 display messages of up to 16 characters each. Each of the 64 messages must have a unique last character, since only the last character is transmitted by the VIP-100. After the messages have been read from paper tape or typed, the VIP-100 is trained for a particular voice. This process consists of the VIP-100 prompting the person by displaying a message, whereupon the person speaks the word or phrase into the microphone. Usually this is done 5 to 10 times for each of the messages. After all the messages have been trained for the person's voice, a recognition phase is entered in which the person speaks any of the 64 messages, and the VIP-100 displays the recognized message data on its 16 character display screen. If the utterance is not recognized, the reject light will be turned on. Individual messages may be changed or retrained, and the final acceptable message data and training data may be output on punched tape for later use.

Every time an utterance is received and recognized during the recognition phase, a character is output to the PDP-10 via the PDP-15. This output character is the last character of the 16-character display message. An inverse process in the PDP-10 program receives the character and retrieves the associated utterance.

Table 2 contains a list of the utterances currently used in CBC demonstrations. We have found this preliminary voice input system to be quite adequate for our current demonstrations.

The eventual speech recognition system employed by the CBC system will incorporate future advances in machine recognition of natural language, using acoustics, syntax, semantics, discourse analysis, and so on. Meanwhile, a few interim improvements are planned in the current VIP-100 system.

We will develop the capability to store message and training data in the PDP-10 and transfer it (via a PDP-11) to the VIP without having to use paper tapes. This will make it possible to have a larger input vocabulary, with certain key words indicating which group of 64 words to use to recognize certain utterances. This will also make it possible to have training data for many persons available simultaneously, perhaps with automatic selection based on recognition of a few sample utterances.

22	SHOWMETHE	(CONTROL Y)	GRAPHICS
23	TABLE	W	GRAPHICS
24	TANK PLATFORM	X	GRAPHICS
25	INSTALL	Y	
26	REMOVE	Z	
27	CONNECTED	1	
28	POSITIONED	2	
29	LASER	3	
30	CAMERA	4	
31	DEDUCE	5	
32	STOP	6	PROC NET & GRAPHICS
33	OBJECTCENTER	7	GRAPHICS
34	CENTEROF	8	GRAPHICS
35	WHATIS	9	GRAPHICS
36	WHATS	0	GRAPHICS
37	VISCENTER	A	GRAPHICS
38	BELT	B	GRAPHICS
39	PRESSURESWITCH GAUGE	C	GRAPHICS
40	CONSULTANT	D	
41	POWERCORD	E	
42	PRESSURESWITCH COVER	F	GRAPHICS
43	EXPERT	G	
44	POINTLASER	I	GRAPHICS
45	WHEREISTHE	J	GRAPHICS
46	PAUSE	K	PROC NET & GRAPHICS
47	BREAK	L	PROC NET & GRAPHICS
48	ROOM	N	GRAPHICS

49	HALT	O	PROC NET & GRAPHICS
50	START	P	
51	PUMP TOP	Q	GRAPHICS
52	PUMP BOTTOM	R	GRAPHICS
53	TANK LEFTLEG	S	GRAPHICS
54	TANK RIGHTLEG	T	GRAPHICS
55	APPRENTICE	U	
56	PUMPBRACE	V	
57	TANK PLATFORM	!	GRAPHICS
58	TANK CYLINDER	#	GRAPHICS
59	CYLINDER	$	GRAPHICS
60	GRAPHICS	&	PROC NET
61	QUIT	.	GRAPHICS
62	CONTINUE	=	
63	PLEASE STOP	↑	PROC NET & GRAPHICS

----

2. Implementation of Voice Output Using VOTRAX Phoneme Synthesizer

The CBC system uses voice output when interacting with a user/technician. For these purposes, we use a VOTRAX model VS-6 phoneme synthesizer, which is a device capable of producing sounds corresponding to 63 phonemes, each with a choice of four inflection values. Phoneme and inflection are under the control of

131

the CBC program. In addition, there are manual controls on pitch, volume, and speech rate. Finally, the VOTRAX itself contains a built-in analysis facility that employs one phoneme look-ahead in order to make the continuous speech sound more natural and intelligible. The result is a mechanical, but quite understandable, voice to use for output in the demonstraton programs.

The VOTRAX consists of a keyboard, a control unit, and a speaker. The keyboard is used for manual input of phonemes, but is not used when the VOTRAX is under computer control. The VOTRAX is attached to the PDP-10 through the teletype patch panel. The computer program sends special 8-bit codes that are converted by the VOTRAX control unit and output through the speaker. (A future modification to the VOTRAX will allow it to share a teletype line with an ordinary computer terminal. This will be an added convenience, but will not change the operation of the program.)

The current use of the VOTRAX is for output of prestored utterances (words or phrases). Each word is represented phonetically, and this is an interesting problem in itself. The VOTRAX is designed to reproduce Mid-Western or standard American English dialect, which is the dialect used almost exclusively by the nationwide media. Still, it is a skill to represent words phonetically (and it is a matter of opinion as to how well a given word has been represented!).

Table 3 contains a listing of the 63 phonemes with an example or description of the associated sound. It should be noticed that some typical English sounds (long vowels, for example) require two phonemes to reproduce them. Some of these are listed in Table 4.

Table 3: VOTRAX Phonemes and Eight-Bit Codes

Phoneme Symbol	Octal Code	Typical Use or Description
PA0	03	(pause)
PA1	76	(shorter pause)
A	40	(to get long A sound use A,AY)
A1	06	(shorter than A)
A2	05	(shorter than A1)
AE	56	hat
AE1	57	(shorter than AE)
AH	44	hot
AH1	25	(shorter than AH)
AH2	10	(shorter than AH1)
AW	75	awful
AW1	23	(shorter than AW)

AW2	60	(shorter than AW1)
AY	41	(to get long A sound use A,AY)
B	16	bob
CH	20	match, chair (CH is T,CH or DT,CH)
D	36	day
DT	04	butter
E	54	(to get long E sound use E or I3,E1)
E1	74	(shorter than E)
EH	73	ten
EH1	02	(shorter than EH)
EH2	01	(shorter than EH1)
EH3	00	(shorter than EH2)
ER	72	her
F	35	fire
G	34	get (not the G in George)
H	33	hay, ahead
I	47	kit (to get long I sound use AH,E1)
I1	13	(shorter than I)
I2	12	(shorter than I1)
I3	11	(shorter than I2)
IU	66	(to get long U sound use IU,U)
J	32	jet, George (J is D,J or DT,J)
K	31	key, sick, car
L	30	lie, well
M	14	my
N	15	nine

NG	24	bring
O	46	(to get long O sound use UH1,O1)
O1	65	(shorter than O)
O2	64	(shorter than O1)
OO	27	book
OO1	26	(shorter than OO)
P	45	pot
R	53	area
S	37	see
SH	21	shy
T	52	tea
TH	71	three
THV	70	then (the voiced TH)
U	50	(to get long U sound use IU,U)
U1	67	(shorter than U)
UH	63	but
UH1	62	(shorter than UH)
UH2	61	(shorter than UH1)
UH3	43	(shorter than UH2)
V	17	seven
W	55	won
Y	51	Mary
Y1	42	(Y used as a consonant as in "yes")
Z	22	zero
ZH	07	azure (the Z)
		measure (the S)

Table 4: Sounds Formed With Two Phonemes

Sound	Typical Use	Phoneme representation
J	jet	D,J or DT,J
CH	match	T,CH or DT,CH
OW or OU	cow	AH1,O1
OI or OY	noise	O1,EH3,E1
Q	queer	K,W
A	name	A,AY
E	tree	E or I3,E1
I	high	AH,E1
	height	AH2,E1
O	note	UH1,O1
U	two	IU,U

The octal phoneme code is six binary bits of information. These are preceded by two bits representing the inflection level. In practice, the inflection code is represented as eight bits (with the last six being zero), and the code sent to the VOTRAX is the sum of inflection and phoneme codes.

Inflection Levels are

1	300	(lowest)
2	200	(normal/default)
3	100	
4	000	(highest)

The null code, which is used for marking end of transmission to the VOTRAX, is the octal code 377 (binary 11111111).

To date, the CBC project has used approximately 800 phonetic words. Many were copied from a list supplied by the VOTRAX manufacturer, and nearly the same number were input by various users of the equipment. A LISP program has been written to handle the storing, retrieving, and transmission of phoneme representations. A few typical representations are reproduced below.

```
(PLEASE (2 P)(1 L)(1 E)(1 E1)(2 Z))
(ASSEMBLE (2 UH1)(2 S)(2 EH1)(2 M)(2 M)(2 B)(2 UH3)(2 L))
(AIR (2 AE)(1 ER))
(COMPRESSOR (2 K)(2 UH1)(2 M)(2 P)(2 R)(2 EH)(1 S)(1 R)(1 R))
```

Adding to the dictionary is simple, via the LISP function ADDW. An example is explained below.

```
ADDW(TESTING)
```

This line is typed by the user.

```
TESTING:
```

This line is typed by the program which then waits for the user to respond with the list of inflection and phoneme pairs.

```
2T 2EH 3S 2T 1NG 1UH3
```

The user inputs a try at the phoneme representation. The VOTRAX pronounces the input string, then the program types OK? and waits for user response.

OK?

The user has a choice of four possible responses.

Y   If the user types Y, the word will be accepted in its present phonetic form.

R   If the user types R, the word will be spoken again.

N   If the user types N, the word is not acceptable and a new list of phonemes must be input.

E   If the user types E, the word is not acceptable but the user is allowed to edit the previous list of phonemes.

This process recycles until the user types "Y", at which time the word is added to the dictionary list and stored in the special data space described below.

The storage of phoneme representations of words makes use of LISP's ability to swap data into an inferior fork. In this case, we use the fork into which LISP swaps compiled code. The space for the phoneme data is a large swapped array. Phoneme and inflection pairs are stored four to a word, and a pointer is kept of the address of the last phoneme stored. When a new word is to be stored, the pointer is incremented by 1, the value of the pointer is stored in the system hash array in the hash address for the word, and the 8-bit codes are stored with an additional leading "0" (making 9-bits each) sequentially into the swapped data array. The last code stored contains a "1" in the left most bit, signaling that the end of the word representation has been reached.

When a word is to be spoken, the "handle" to the phoneme representation is obtained by GETHASH(word) and the list of phoneme codes is retrieved from the swapped data space and put into a buffer (a LISP array) to be sent to the VOTRAX. The capacity of the buffer is 34 words or 136 phonemes. A pointer is kept to the current end of data in the buffer, and many words and phrases may be stored into the buffer before they are actually transmitted. Transmission to the VOTRAX (across the teletype lines by the TENEX operating system) takes place either when the buffer is filled, or when the program determines that no more words are to be utterred. The end of transmission is signalled to the VOTRAX by placing a code "377" at the end of the data in the buffer. The VOTRAX itself has a buffer. Voice output is initiated by the VOTRAX when its buffer is half full. Thus, it is possible to send a second transmission to the VOTRAX before the buffer is completely empty, making continuous speech possible.

Provision is made for saving and restoring dictionaries so that phoneme representations must only be input once. The LISP functions MAKEDICT and LOADDICT accomplish this task. They depend on a list of words being saved as the value of the LISP atom called "DICT". (In demonstration programs, we frequently set DICT to NIL to save the space required to link all the dictionary words into a single list. This means, however, that we cannot save an updated version of the dictionary; but this is not usually a hindrance.) In the case of MAKEDICT, a disk file is opened, and each word on DICT is written on the file with its corresponding phoneme list (not the binary form--rather the form that explicitly contains inflection numbers and phonemes). LOADDICT operates in reverse, opening a specified disk file for input, reading sequential entries, converting phonemes and inflections to binary codes, and storing them in the swapped data area.

The use of computer voice output has been very satisfactory in all the demonstrations we have conducted thus far. We intend to continue to use the VOTRAX for the forseeable future. Refinements will be made in the representations of existing words and new words will, of course, be added. A program will be written that will attempt to make its own phonetic representation of a new word. Another new program will provide for dynamic inflection of a whole utterance, which will be a great improvement over the current method of prestoring the inflections for each individual word.

F. System Integration

The ultimate form of the Computer Based Consultant is expected to be a single computer program which may be, in fact, a combination of many different, but interlinked, computer programs. The system building task is a big and important one and there are few precedents or theories for how to do it. The main philosophy that has guided system building of the CBC up to now is that as different programs and "abilities" become available (even if they are only in preliminary or simple form) they are included in the system as soon as possible. This approach has several advantages.

The main advantage is that we can get a "flavor" of what the eventual system will be like. For example, an early system was just the procedural net program, expanded with air compressor semantics and knowledge about tools. It soon became apparent that voice input and output would be "nice", so that was added by taking advantage of available devices that use preselected words and phrases for input and output.

Things that are seen as deficiencies by casual observers of the demonstrations are taken as important next steps for the system builders. For example, it soon became apparent that many questions an apprentice would logically ask, such as why?, where?, what if ...? , and so on, could not be handled. The mechanisms for a few of these were coded in rudimentary form so that their effects could be tested.

Another important consequence of continual system building is learning where different parts of the system overlap. For example, several different "models" are already in use (i.e., a connectivity model for deductive question answering, and a polyhedral model for graphical display and pointing). We must ask whether the models can be combined. Are they compatible? Can all models be kept up-to-date simultaneously without too much trouble? Where can one program get information from another program's model?

In summary, the current (April 1975) system has the following capabilities:

(1) It knows about top level components of a specific air compressor.

(2) It knows about some basic hand tools (screwdrivers, wrenches, allen-wrenches) and fastenings (screws, machine-screws, nuts and bolts, set-screws, washers).

(3) It knows about assembly and disassembly sequences (positioning, connecting, fastening, removing, extracting).

(4) It knows about parts adjacent to other parts, both in the connectivity (assembly/disassembly) model and in the polyhedral (graphics) model.

(5) It can use a laser beam to point at the location of a part.

(6) It can determine which part is indicated by a light/pointer or laser beam.

(7) It can use voice input and output in a simple form.

(8) It can create plans for accomplishing tasks and answer questions about the plans.

(9) It can execute the plan (i.e., tell a user how to perform a task at levels of detail based on the user's varying responses).

(10) It can answer questions about the changed state of the equipment at any time.

(11) It can perform simple execution monitoring of the progress of the apprentice in executing a plan.

(12) It can perform calibration of the laser-rangefinder and the TV camera in a semiautomatic mode.

This is an impressive list of capabilities for a program that is still in its early stages. Each of the capabilities needs more development, and certain important capabilities are included only in rudimentary form (such as execution monitoring, natural language, and vision). Still other capabilities such as troubleshooting are not included at all. Yet, by making an integrated system out of available parts we are able to envision the final system much more readily and recognize where our efforts should be directed.

The "top level executive" or driver program for this system is expected to be the execution monitoring portion of the procedural net (NOAH), because this is the program that carries on the basic interaction with the user. This may be closely associated with the eventual speech understanding program, with the semantic net model becoming one of the primary sources of information about the task (and world) domain.

The system is currently implemented by taking advantage of the fork structure under the TENEX operating system. The main programming languages are LISP and QLISP, with use of SAIL, FORTRAN, and machine language where appropriate. The fork structure is shown in Figure 16.

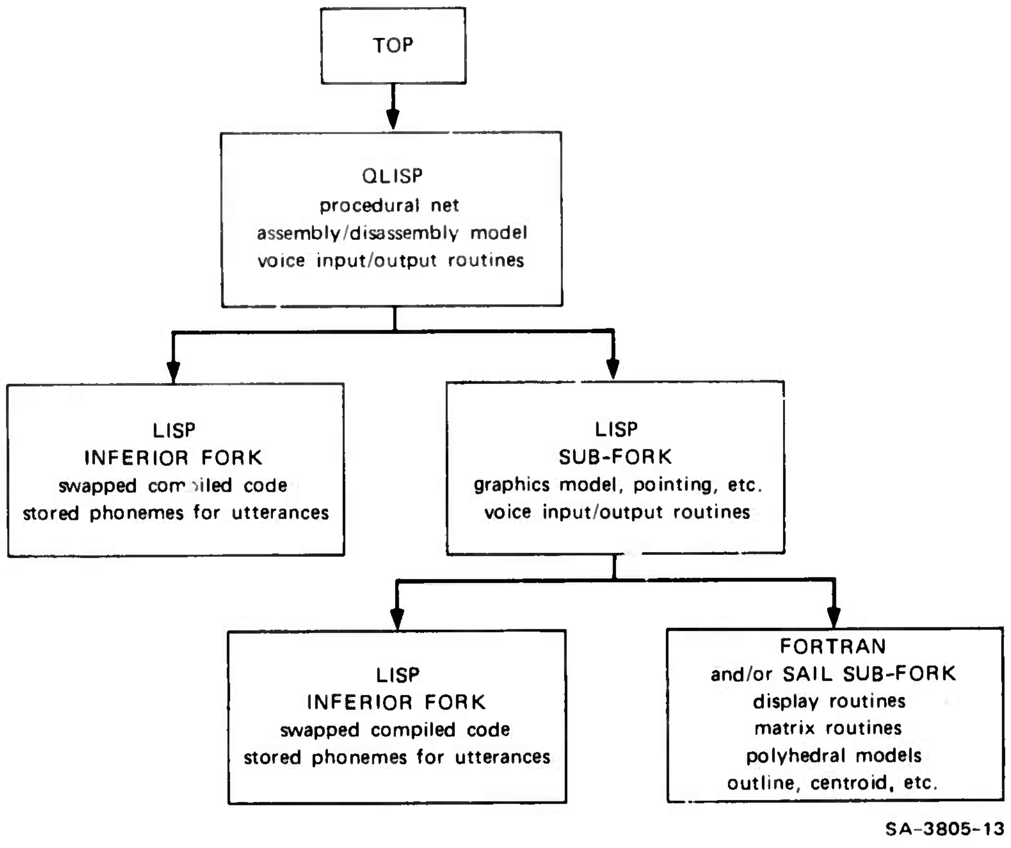


FIGURE 16 STRUCTURE OF THE CBC DEMONSTRATION PROGRAM

III. COMPONENTS OF SUBSEQUENT SYSTEMS

A. Introduction

In parallel with constructing the present system, we have put in a great deal of design work on components of future (1976 through 1978) CBC systems. In Section III.B we present a detailed description of our work to provide a powerful natural language understanding system. Our intention is that beginning in 1976 we will be able to process some text input to the system. (The text input system is being designed to be part of a system that will ultimately handle speech.) During 1977 we plan to make extensive additions to the power of the text system, and finally in 1978 we expect to extend it to handle utterances spoken by the apprentice. Gary Hendrix and Barbara Deutsch have been responsible for our natural language research working in conjunction with the SRI speech understanding project.

In Section III.C we describe two pieces of additional problem solving work. First we present some research performed by Marty Rattner on how to model and compute the "freedom of movement" properties of equipment. Next we discuss the preliminary results of Richard Duda and Nils Nilsson toward building a system for diagnosis of equipment malfunction.

Our work in vision is described in Section III.D. It will ultimately provide the capability of answering questions about the equipment and the workstation by direct observation. The vision work represents the combined efforts of several people, notably Marty Tenenbaum, Harry Barrow, and Thomas Garvey (doing Ph.D. dissertation work).

B. Natural Language

1. Introduction

The goal of our work in language is to provide a two way communications link between the consultant and the apprentice that is both natural and convenient. We are persuaded that the most appropriate communications medium is that of spoken English. The reason for this is twofold. First, speech frees the apprentice from the burdens of interacting mechanically with the consultant while performing his tasks. Second, the use of English makes it unnecessary for the apprentice to cope with a formal language, thus allowing him to concentrate all his efforts on the performance of the work task. It is anticipated that by 1978 the language component will have the ability to understand the user's spoken questions and remarks regarding the state of his task and the use of tools. Further, the system will have the ability to produce spoken responses to verbal inputs.

In developing a technology for the understanding of continuous speech, our research efforts are being carefully coordinated with those of the SRI speech project [36]. As a result of cooperation during the past year, a common basic grammar and a common semantic interpretation system have been developed. Further, the focused parser designed for the speech project has been adapted for (interim) text processing by the CBC. While various acoustic processing techniques are being developed, this parser will allow us to conduct experiments during the next two years using text and will facilitate the eventual conversion to acoustic input in 1978.

In keeping with the goal of providing natural communications between consultant and apprentice, we have, during the last year and a half, collected and analyzed numerous protocols between human consultants and apprentices. Within the past year, this analysis has aided us in the selection of a working vocabulary and in the development of a tentative performance grammar (which extends the grammar developed jointly with the speech project). Having used the protocol experiments to clarify the language requirements of a computer based consultant, we have been developing a system architecture and a semantic representation scheme capable of dealing with the linguistic problems likely to arise in consulting situations. Details concerning each of these achievements follow.

2. Protocol Collection

We have extended the collection of protocols described in our previous annual report [37] to include different tasks and apprentices at different skill levels. One new task has entailed working on a subcomponent of the compressor, namely the pump. This task introduced new vocabulary and, more importantly, the need to describe and understand questions about more complex tools, parts, and operations. A sample from one of the pump assembly dialogs is shown in Figure 17.

We have an experimental setup now that allows us to record new protocols easily. We plan to run additional experiments on new pieces of equipment. Tasks for these experiments will entail troubleshooting as well as assembly and disassembly.

3. The Concordance Programs

As an aid to our analysis of the language used in expert/apprentice situations, we have developed a package of concordance programs. A sample output from these programs, illustrating several of the features of the concordance produced, is shown in Figure 18. This sample shows a collection of results from concordances done separately for several dialogs. (The dates were added to make the separation clearer.) For each of the that which occur, the concordance shows a frequency count and a list of the different contexts in which that word appears.

```
APPRENTICE:OK,  WHAT NEXT?
00:59:44

00:59:51
CONSULTANT:PLACE THE CYLINDER HEAD GASKET ON THE TOP OF THE CYLINDER
01:00:12

01:00:27
U:WHAT DOES THE CYLINDER HEAD GASKET LOOK LIKE?
01:00:36

01:00:38
C:THE CYLINDER HEAD GASKET IS A SQUARE PAPER SHEET WITH A LARGE ROUND
  HOLE IN THE CENTER, AND 4 BOLT HOLES ON THE OUTSIDE.
01:01:31

01:01:44
U:OK, I FOUND IT. I'VE PLACED IT.  WHAT NEXT?
01:01:52

01:01:54
C:INSTALL THE CYLINDER HEAD.
01:02:02

01:02:18
U:WHICH WAY DOES IT GO?
01:02:22

01:02:26
C:THE INTAKE BREATHER SHOULD BE ON YOUR RIGHT AS YOU FACE THE
  THE CRANKSHAFT.
01:03:21

01:03:45
U:OK, IS THIS RIGHT?
01:03:49

01:04:02
C:NO, YOU HAVE IT BACKWARDS.
01:04:09

01:04:22
U:WHAT TOOL DO I USE TO ATTACH IT?
01:04:28
```

FIGURE 17   SAMPLE PUMP ASSEMBLY DIALOG

```
@ALL@ 3/1893 0 %      (FEB12)
                         09900 THE SETSCREWS OR DO I HAVE TO TAKE THEM     ALL     OUT?
                         22500 CE I HAVE GOT NY FINGERS UNDER THERE AT     ALL     THE BOX WRENCH
                         24600     A;JUST FOR YOUR INFORMATION IVE GOT     ALL     FOUR BOLTS IN PLACE MJUMBLE

@ALL@ 3/1033 0 %      (OCT30)
                         05600                               C;OK,     ALL     RIGHT.
                         23000                C;DOES THE PISTON GO     ALL     THE WAY INSIDE THE CRANKCASE?
                         47600                      C;OH? THAT'S       ALL   ?  WHAT ABOUT THIS PART HERE? OH, TURN OFF

@BOLT@ 4/1627 0 %     (FEB13)
                         05600                      A;WHAT'S A MOTOR   BOLT
                         21200                           A;UH ONE      BOLT     IS IN AN AWKWARD PLACE
                         40700          A;IM TIGHTENINNG THE SECOND    BOLT
                         41300       A;IM IM STRAIGHTENING MY LAST     BOLT

@BOLTED@ 1/1627 0 %   (FEB13)
                         20700 OLTS FROM THE BASE OF THE PUMP THAT ARE     BOLTED      ONTO  THE

@BOLT@ 3/1611 0 %     (MAY24)
                         05100                                  C;     BOLT     THE PUMP TO THE BASE PLATE
                         07100                            HOLD THE     BOLT   , AND ONE UNDERNEATH TO TIGHTEN THE NUT
                         37300                          C;Fine one     bolt     is on and three more have to go.

@BOLTED@ 2/1611 0 %   (MAY24)
                         12400                             U;it is     bolted    , now, what should I do?
                         38100 he pulley is in place now; the motor is     bolted     on.  what

@BOLT@ 4/1148 0 %     (NOV12)
                         50800                        DEGREES, AND     BOLT     IT ON
                         52000                                  C;     BOLT     THE BEARING PLATE ONTO THE CRANKCASE
                         55200                                  C;     BOLT     THE BOTTOM PLATE TO THE PUMP
                         62400                                  C;     BOLT     THE CYLINDER HEAD  TO THE PUMP HAVE YOU I

@BOLTED@ 1/1148 0 %   (NOV12)
                         63700                           C;I HAVE      BOLTED     THE CYLINDER HEAD TO THE PMP.

@GET@ 5/1627 0 %      (FEB13)
                         06400        A;WHICH TOOLS SHOULD I USE TO    GET     THE BOLTS THAT ARE HARD TO UNSCREW
                         07100                  AND THE WRENCH TO      GET     IT UNSTUCK BUT I HAVEN'T HAD MUCH LUCK
                         13600        A;TELL UM I FOUND AN ANGLE I CAN GET     AT IT
                         26300  MY FINGERS AREN'T QUITE LONG ENOUGH TO GET     ONE OF THE BOLTS  BACK
                         29700                          A;OK HOW DO    GET     THE WHEEL FORCED BACK ON

@GET@ 4/914 0 %       (FEB20)
                         02700              A;USING  THE  PLIERS TO    GET     THE NUTS IN UNDERNEATH THE PUMP
                         03900   YOU  HAVE  AN ALTERNATE BETTER WAY TO GET     THE NUTS IN
                         17000                              ONE TO     GET     AT.
                         19800 T OUT THEN IT WOULD HAVE BEEN EASIER TO GET      AT  THEM  AND
```

FIGURE 18   CONCORDANCE SAMPLE

Knowing the context in which a word appears aids in determining both the different semantic and the different syntactic uses of the word. We have used this information in constructing our working concepts and our working grammar. By pulling all of the uses of a word together, the concordance enables us to determine which word senses are used most often. It would be difficult to get this same information from the raw dialog because the word uses are so spread out. As an example, consider the uses of the word "all" shown at the top of Figure 18. These are representative of the uses we found. In only one case is "tall" used in the quantificational sense (in the FEB12 dialog, ". . . I've got all four bolts in place"). This result (and similar ones for the other quantifiers) was unexpected.

The number to the left of each entry in the concordance is the line number in the dialog where this entry appears. This allows us to examine the total context of the word use. We have used this to distinguish expert and apprentice word uses. For example (see Figure 18) the use of "bolt" as a present tense verb comes exclusively from the expert. The apprentice uses the verb form of bolt only in the past tense.

It is possible to merge the concordance results for several dialogs. We have obtained a merged concordance for all of the dialogs, which enables us to distinguish individual differences from general trends. Those words and syntactic structures that appear across several dialogs are incorporated in our working vocabulary and working grammar. Later, individual modes of expression will be handled in the user model.

4. Dialog Analysis

As mentioned previously, a major motivation in collecting the dialog protocols has been to determine the language requirements of the CBC. Analysis of the collected dialogs has proceeded on several levels. On the most basic level, we have used the protocols in establishing our working vocabulary and grammar. These are discussed more fully in separate sections. In both these undertakings, analysis of the dialogs aided us not only by illustrating what language usages there were but also by making us aware of what problems did not arise. For example, as noted before, we were surprised by the infrequent use of quantification by the apprentices. Quantifier words, like "all", did appear in apprentice statements, but mainly in phrases like "is that all" meaning "have I finished?".

There are many well known discourse level problems in natural language understanding; for example the problems of resolving reference and completing partial utterances. In the CBC domain there are also the closely related problems of describing and understanding descriptions of objects and actions. We have been using the dialogs as guides for our research on these more global discourse level problems.

We have been interested not only in the forms of these CBC discourse level problems, but also in discovering what inherent information there was in the dialog context that would aid in "solving" the problems.

The dialog context is actually a composite of three different component contexts: a verbal context, a task context, and a context of general world knowledge. The verbal context includes the history of preceding utterances: their syntactic form, the objects and actions discussed in them, and the particular words used. The task context is the focus supplied by the task being worked on. It includes such things as: where does the current subtask fit in the overall plan, what are its subtasks, what actions are likely to follow, and what objects are important. The context of general world knowledge is the information that reflects a background understanding of the properties and interrelations of objects and actions; for example, the fact that tool boxes typically contain tools and that attaching involves some kind of fastening.

An important aspect of the reference problem is determining what sources of knowledge should be accessed to resolve a reference. Decisions must be made concerning how much effort may be spent testing one antecedent candidate and how much effort may be spent investigating the different context perspectives from which that candidate may be viewed.

In the context of

"Tighten the setscrews with an allen wrench"

consider the question

"Where are the setscrews?"

The phrase "the setscrews" must be resolved as the previously mentioned ones. This resolution comes from the verbal context (or dialog history). However, in the context

"Attach the pump pulley next"

the question "Where are the setscrews?" can only be understood if the consultant is aware that installing and tightening some screws is part of the operation of attaching the pump pulley. The resolution comes from knowledge of the task. Any screws mentioned in the previous dialog would probably be irrelevant. Finally, if we consider the context

"I have the parts box,"

the resolution can be found only by knowing that screws are typically stored in a parts box.

The reference resolver must consider as candidates for antecedent not only objects and actions that are explicitly represented in the dialog history (which would work only for the first of our examples), but also the interconnections of those objects and actions in the task domain and in the "general real world". This means deciding which kinds of connections to consider first and how long to investigate them before looking at others. It also entails deciding how much effort to put into looking at all the connections of one object or action before considering others. The implementation of the reference resolver is being designed so that we can easily experiment with different strategies for looking at the various contexts. The separation between the three context components is made explicit. The task context is supplied by a connection to the procedural net. The difference between the local verbal context and real-world knowledge connections is reflected in the way the semantic representation of the discourse history is kept. Essentially, the local context is kept separate from global knowledge, but contains a few links to that knowledge. How this is done will be discussed more fully after the semantic representation is presented.

The problem of object description is closely related to the reference problem. This is obvious since the description problem is basically the inverse of the reference resolution problem: an object is unambiguously described if the description given can be used to locate a unique object. Any object has a multitude of attributes. Some are simple (e.g., color, shape) and others involve connections to other objects (e.g., on-top-of, inside). However, at any one time only a few of these properties are needed to uniquely specify the object. This is because context limits the other objects from which any one object needs to be distinguished. As an example, consider the situation when the apprentice is using a 1/2-inch box-end wrench and a 1/2-inch socket wrench to tighten a nut/bolt fastening. The two wrenches may be distinguished by type: "the wrench" is ambiguous, but both "the box-end wrench" and "the socket wrench" are unambiguous. However, if the apprentice is using two 1/2-inch box-end wrenches for his task then they need to be distinguished by other criteria, such as which is on the nut.

We have been designing the discourse history component of the system in a way that will allow us to know which attributes of an object are most important in a given context. Again this is closely tied to the semantic representation and will be discussed in Section 9 below.

5. Selection of Working Vocabulary and Working Concepts

The concordance programs have aided us in finding which words are likely to occur in the workstation. Surprisingly, only 620 different words were used in all of the dialogs. Our initial working vocabulary of approximately 650 words includes some additional words we know will appear as new tasks are added (e.g., some part names).

As a first step toward building a semantic representation of the knowledge base necessary to understand CBC related dialogs, we have divided the vocabulary into conceptual categories. (Our semantic representation is discussed in Section 8 below. For present purposes, it suffices to know that objects and actions are represented by nodes in a semantic network. Various hierarchical relationships between categories of objects and actions are also represented. In addition, it is important to know that the result of parsing an utterance is the production of a semantic network.) At present there are 21 conceptual categories. The categories, along with a sample of the words in each category, are shown in Figure 19.

There are several characteristics of this category list that will be important in building our network representation. First, the categories are hierarchical. For example, in the category of location operators there are general position change words (e.g., adjust) and also position change words which specify that a tool be used (e.g., pound). Note that some of the hierarchies that are important in the CBC domain are not shown in this category list, because subclasses are not shown to all levels. (To do this would require showing the complete semantic net). For example, under tools we do not show the hierarchy tool→ wrench→ box-end wrench.

Note that some of the categories are quite closely related. The best example of this is the set of categories: location, location operators, locate. The location category contains

FASTEN/UNFASTEN	
(un)fasten - general	attach,install,remove,undo
(un)fasten - specific	bolt,screw,unplug,unscrew
fasten-sets	assemble,replace
fasten/unfasten -weakobs	loosen,tighten
PHYSICAL OBJECTS	
comparts	aftercooler,pump,motor,flywheel
tools	box-end wrench,pliers, wheelpuller
tools-parts	extender,joint
objs -environ	table,toolbox,workbench
parts-other	machine,part,piece
MATERIALS	
materials-solid	copper,plastic,steel
materials-fluid	air,oil
PHYSICAL FEATURES	
surface-features	burr,keyway,rim
shapes	
shapes	band,disk,semicircle
shapes-adj	dish-shaped,hexagonal,round
shapes-props	convex,flat
colors	gold,green
how-made	cast,forged,machined
part-func	cooling fins,protective
STATE(miscellaneous)	
state	bent,new
state-change	break,file
LOCATION	
places	base,floor,top
position	above,inside,there
position-descript	askew,flush,vertical
LOCATION OPERATORS	
position-change	
position-change	adjust,position,put,take-off
position-change-rotation	rotate,turn
position-change-tool	hit,pound
position-maintain	
position-maintain	balance,hold,stand
position-maintain-tool	hold-with
LOCATE	
locate	find,lose
CONTAIN	
contain	contain,escape,leak

FIGURE 19 CONCEPT CATEGORIES

Reproduced from best available copy.

```
PRESSURE
pressure                          pressure,strain
pressure-verbs                    build-up,rise

SCALERS
attribute-mods                    enough,excess,only
comparisons
  comparisons-degree              as---as,more,slightly
  comparisons-effort              difficult,effort,tedious
  comparisons-phys-other          light,shiney
  comparisons-similarity          act-as,like,same
  comparisons-size
    comparisons-size-bulk         big,small
    comparisons-size-linear       long,thin
manner                            firmly,gently
constraint                        evenly,freely,tight

MEASURE
measure-accuracy                  approximate,exactly
measure-portion                   half,portion,rest
measure-sizedimen                 angle,radius,size
measure-units                     degrees,minute,ounce

TIME
time                              time
time-comparisons                  ahead,before,next
time-duration                     still,until,while
time-repeat                       again,never,sometimes
ordinals                          first,second

PROCESS
process                           happens,task
operate                           operate,work

PROGRESS
progress                          begin,done,progress,try

DATA
data                              idea,instruction

PEOPLE
people                            apprentice,consultant,I,you

COMMUNICATION
verbal                            answer,message,tell
visual                            picture,point,see
touch                             feel
auditory                          hear,noise,sound
exclamations                      aha,oh,oops
polite                            please,thank-you
yes                               correct,fine,ok,yeah,yes
warn                              be-sure,note,warning
```

FIGURE 19 CONCEPT CATEGORIES (Continued)

Reproduced from best available copy.

ENABLE	
ability	able,can,in-order-to
advise	advice,suggest
cause	make
need	need,require
MENTAL CONCEPTS	
knowledge-state	assume,certain,think
understand	ok,realize,understand
identify	call,identify,point
decision	decide,determine,test
LINGUISTIC OPERATORS	
articles	a,the,these
auxiliaries	be,do,have
conjunctions	also,and,but,or
modals	can,must,perhaps,should
not	not
prepositions	by,for,to,together,with
quantifiers	all,both,either,some
subordinate conjunction	because,if,then,when
wh?	what,when,which,why
pronouns	it,that,they,thing
proverbs	use
^L	

FIGURE 19 CONCEPT CATEGORIES (Concluded)

those words that express the different places and positions in which an object may be found. The location operators are those words that have to do with effecting change (or no change) on those positions. The words in the locate category have to do with whether or not the location of an object is known. The representations corresponding to the words in these categories will be closely linked.

In addition to the categories that have to do with objects (e.g., tools) and actions (e.g., fasten) in the domain, there are categories that have to do with expert/user interaction (those in

Reproduced from best available copy.

the communication category). The presence of this category indicates the importance of the subtask of communicating information between the the expert and the apprentice.

There are also categories that in essence concern the syntactic structure of individual utterances. The linguistic operators category contains that which will not themselves have specific representations in the semantic net. The words in this category help guide the construction of the semantic representation being built up for an utterance, not by adding new semantic constituents, but rather by indicating the relationships between existing constituents. For example, the prepositions help supply the mapping between surface cases and deep conceptual cases.

6. The Working Grammar

Through cooperative efforts with the SRI speech project, a basic performance grammar applicable to a wide variety of tasks has been developed. For use in the CBC domain, certain extensions to this basic grammar have been made to account for syntactic constructions that occur with higher frequency in consultation dialogs than in ordinary conversations. The grammar is based on a thoughtful analysis of our protocol experiments, and thus reflects a major portion of the language actually spoken by apprentices in the performance of tasks rather than some abstract theory of how the apprentice "should" speak. Thus, the grammar can handle isolated phrases and clauses as well as complete sentences. Although it reflects the language of this task domain, the grammar does include the kinds of linguistic constructions handled in contemporary computational linguistic research and was written with extensibility in mind.

Since the grammar is ultimately to be used in the processing of speech, it has been designed so that advantage may be taken of knowledge available from whatever sources are capable of reporting information at any given point during an analysis. The grammar is written so that it is not restricted to a particular kind of parsing strategy. That is, it can be used both bottom up in building more complex grammatical structures from words that have been identified acoustically, and top down in working from predicted grammatical structures to the words they contain. (The same grammar should also be serviceable for response generation, although not much study has yet been made in this area.) The grammar incorporates in its rules information on semantic, pragmatic, and prosodic features, as well as on grammatical ones. Experiments conducted by the speech group [36] show that when the basic grammar is compiled with a lexicon into the internal representation used by the parser, the result is a language processing system whose sources of knowledge are highly integrated and well coordinated.

Both the basic grammar and the CBC-specific extensions are expressed as collections of rules (following Paxton

[38]). As exemplified by (simplified and edited) rule S1 of Figure 20, a grammar rule has three principal components: a context-free production, an attributes statement, and a factors statement. While each rule contains a context-free component, it must be emphasized that the rules themselves are not context free, since the application of a production is constrained by restrictions encoded through the attributes and factors statements.

```
RULE.DEF  S1:
     CF-PRODUCTION
               S = NP VP;
     ATTRIBUTES
          FOCUS, MOOD FROM NP,
          VOICE = "ACT,
          SEMANTICS = SEMRS1(SEMANTICS(NP),
                             SEMANTICS(VP)),
          AFFNEG = IF MOOD EQ DEC THEN "AFF;
     FACTORS
          IF VOICE(VP) EQ PASS THEN OUT,
          IF GINTERSECTION(NBR(NP), NBR(VP))
             THEN OK ELSE OUT,
          IF TRANS(VP) EQ 0 THEN BAD,
          IF SEMANTICS EQ NIL THEN OUT;
```

FIGURE 20 A SIMPLIFIED GRAMMAR RULE

The context-free production of a grammar rule specifies a string of utterance constituents (either terminals or tokens) that, under favorable circumstances, may be combined to form a larger or more complex constituent. For example, the context-free production of rule S1 is

S = NP VP

which indicates that an S constituent of an utterance (a complete sentence or clause) may be formed by joining an NP (noun phrase) with a VP (verb phrase); however, various attributes of the NP and VP must be in agreement.

To build up internal descriptions of parsed phrases, each rule augments its context-free production with an attributes statement. Each constituent type (e.g., the constituent types NP, VP, S, U used by the grammar is associated with a set of attributes whose values help to characterize particular instances of the constituent. For example, the attributes VOICE and MOOD help to characterize particular instances of VPs and Ss. The attributes statement of a grammar rule assigns values to attributes of the phrase being built up. These values are derived primarily from the values of attributes in the phrase's constituents. Thus, for example, the FOCUS and MOOD of an S constructed by rule S1 are taken from the FOCUS and MOOD of the NP that is the first constitutent of the S. Attribute-values for the most primitive components of the grammar (such as individual nouns) are specified by the lexicon.

To constrain the application of rules to appropriate situations and to aid the parser in deciding which rules seem most likely to yield good results, each rule contains a factors statement. Since a given context-free production is applicable only if phrase subparts are compatible, the factors statement may apply tests that examine the values of constituent attributes to determine their compatibility. For those phrases that pass the compatibility test, the factors statement derives a number, called the rule score, which indicates how well the constituents fit together. This number is used by the parser in determining how much effort to devote to using the phrase produced by the rule in subsequent computations.

To use a rule such as S1 in the bottom up mode, the parser first finds a string of the form "NP VP" to match the right side of the context-free production. With such a string found, the parser begins to check the factors. First, a check is made to see if the VOICE of the VP is PASS (passive). Passive voice is not allowed in this rule (there is another rule, S9 in Figure 21, that should be used for passives), and hence if VOICE(VP) is in fact PASS, then this application of rule S1 is thrown OUT and a record of the aborted application is made to prevent future attempts. If the voice test is passed, then a test is made to see if the number attributes of the NP and VP are compatible; e.g., "they mesh" is OK but "it mesh" is OUT. The passing of this compatibility test is regarded as a moderately good sign (indicated by "OK") and the rule score becomes slightly enhanced. The next portion of the factors statement checks to see if the TRANS of the VP (the number of yet unassigned standard case arguments) is 0. If TRANS(VP) is in fact 0, this is a BAD sign and detracts from the score given the resulting S constituent.

When other tests are passed, the SEMANTICS attribute of S is computed from the SEMANTICS of NP and VP by a call to the

RULE	CONTEXT-FREE-STATEMENT	
U0	U = "OKAY	cbc
U1	U = S	
U2	U = NP	
U3	U = NOM	
S1	S = NP VP	
S2	S = NP AUXD VP	
S3	S = NP:NP1 AUXB NP:NP2	
S4	S = VP	
S5	S = AUXD VP	
S6	S = NP:NP1 AUXD NP:NP2 VP	
S7	S = AUXD NP VP	
S8	S = AUXB NP:NP1 NP:NP2	
S9	S = NP AUXB VP	
S10	S = AUXB NP VP	
S11	S = NP AUXB ADJ	
S12	S = AUXB NP ADJ	
S13	S = "HOW ADJ AUXB NP	
S14	S = "THERE AUXB NP	
S15	S = "THERE AUXB NP PREPP	
S16	S = NP AUXB "THERE	
S17	S = NP AUXB "THERE PREPP	
S18	S = AUXB "THERE NP	
S19	S = AUXB "THERE NP PREPP	
D1	AUXD = DO	
D2	AUXD = DO "NOT	
D3	AUXD = DO -NEG	
B1	AUXB = BE	
B2	AUXB = BE "NOT	
B3	AUXB = BE -NEG	
NP1	NP = NOM	
NP2	NP = NUMBERP	
NP3	NP = NUMBERP "OF NP	
NP4	NP = NUMBERP NOM	
NP6	NP = DET	
NP7	NP = DET "OF NP	
NP8	NP = DET NOM	
NP9	NP = DET NUMBER "OF NP	
NP10	NP = DET NUMBER NOM	
NP11	NP = DET NUMBER	
NP12	NP = QUANT NUMBER "OF NP	speech
NP13	NP = QUANT "OF NP	speech
NP14	NP = QUANT NUMBER	speech
NP15	NP = QUANT	speech
NP16	NP = QUANT NUMBER NOM	speech
NP17	NP = QUANT NOM	speech

FIGURE 21 CONTEXT-FREE PRODUCTIONS OF BASE GRAMMAR

```
NOM1        NOM = NOMHEAD
NOM2        NOM = ADJ NOM
NH1         NOMHEAD = NOMHEAD PREPP
NH2         NOMHEAD = NOUN
NH3         NOMHEAD = NOMHEAD S
NH4         NOMHEAD = NOMHEAD VP
N1          NOUN = N
N2          NOUN = N -PLURAL
N3          NOUN = CLASSIFIER NOUN          cbc
N4          CLASSIFIER = VP                 cbc
N5          CLASSIFIER = CLASSNOM           cbc
N6          CLASSNOM = ADJ CLASSNOM         cbc
N7          CLASSNOM = CLASSIFIER N         cbc
N8          CLASSNOM = N                    cbc
VP1         VP = VERB
VP2         VP = VP NP
VP3         VP = VP PREPP
V1          VERB = V
V2          VERB = V -SG
V3          VERB = V -PPL             pastparticiple
PREPP1      PREPP = PREP NP
DET1        DET = NP -GEN
NUMP1       NUMBERP = "HOW NP
NUMP2       NUMBERP = NP
NUMP3       NUMBERP = THANR "THAN NUMBER
NUMP4       NUMBERP = NUMBER
NUM1         NUMBER = SMALLNUM
NUM2        NUMBER = SMALLNUM "HUNDRED
                             "AND SMALLNUM
NUM3        NUMBER = SMALLNUM "HUNDRED SMALLNUM
NUM4        NUMBER = SMALLNUM "HUNDRED
NUM5        NUMBER = "HUNDRED "AND SMALLNUM
NUM6        NUMBER = "HUNDRED SMALLNUM
NUM7        SMALLNUM = DIGIT
NUM8        SMALLNUM = TEEN
NUM10       TEEN = DIGIT -TEEN
NUM11       DIGTY = DIGIT -TY
NUM12       SMALLNUM = DIGTY
NUM13       SMALLNUM = DIGTY DIGIT
```

FIGURE 21 CONTEXT-FREE PRODUCTIONS OF BASE GRAMMAR (Concluded)

function SEMRS1. If SEMRS1 returns NIL, the rule fails on semantic grounds. Otherwise, SEMRS1 returns the semantic interpretation of S, and the other attributes of S are computed in accordance with the attributes statement and stored with the completed S.

While space considerations make it impractical to list the complete grammar in this document (see Ref. 36 for a complete listing), a feel for the extent of the grammar may be gained by examining just the context-free productions of the rules. The CFPs of the base grammar are presented in Figure 21, with CBC extensions in Figure 22. (A base rule marked by "cbc" (or "speech") indicates that the rule is currently used primarily by the CBC (SRI speech) project.)

```
RULE     CONTEXT-FREE-STATEMENT

S20      S = NP AUXB PREPP
S21      S = AUXB NP PREPP
S22      S = "WHERE AUXB NP
S23      S = REL S        REL = that, which
S24      S = SEQWD W      SEQuence worD = now,
                                      next, first
S25      S = SEQWD S
S26      S = SUBORDS S
S27      S = S SUBORDS
                 Example:  Turn the bolt
                               until it's tight.
S28      S = S "PLEASE
S29      S = "PLEASE S
SUBS1    SUBORDS = SUBCONJ S    SUBCONJ = when,
                                  until, so that
NP18     NP = NP:NP1 COORD NP:NP2       CORD =
                                       and,or
NP1      NP = NP:NP1 COMMA NP:NP2
NOM3          NOM = NOM:NOM1 "AND NOM:NOM2
```

FIGURE 22 CBC GRAMMAR EXTENSIONS

7. The Parser

The SRI speech group has developed an experimental parser that is specifically designed to cope with the variability of pronunciations and the frequent lack of separations between words that characterize spoken language. To take advantage of knowledge about the input stream from any sources that are available, the parser is allowed to (simultaneously) work both top down and bottom up, predicting words and phrases on the basis of context and building up phrases and clauses from words that have been identified in any position in an utterance. Using information from many sources, the parser coordinates knowledge relating to the structure of English, the nature of the task being performed, the conversation context, the prosodic features of the input, and so on. Since the uncertainty of the input and the variety of kinds of knowledge required can lead to consideration of a large range of interpretations in the analysis of an utterance, the parser contains mechanisms that enable it to examine the most reasonable alternatives first and to focus its activities with respect to both processing time and available space. (A full description of the parser is contained in Ref. 36.)

To facilitate the conversion to spoken input in 1978, we are designing our interim text processor to be as nearly like the proposed speech processor as possible. Toward this end, the speech parser has been modified to accept text input. Although a simpler parser could be written to do text processing exclusively, by using a modified speech parser in the interim, the eventual conversion to spoken input will entail no interface changes between the parser and the CBC-specific systems with which the parser communicates. Both the CBC-specific systems that communicate with the parser and the CBC grammar may be written and tested in text mode. Further, the text-based system will serve as a useful tool for testing task-oriented input-prediction strategies that will be needed to help guide the parser through spoken consultation dialogs.

8. System Design

Since the language system must be coordinated with other components of the CBC, we have attempted to temper our work in protocol analysis and in the development of a semantic representation by keeping in mind the interactions that must occur between the language system and various knowledge specialists in the CBC. To better understand the intercommunications problems, we have developed tentative system designs at two levels.

The most abstract design, shown in Figure 23. interprets the language component as the interface between the user and a group of task specialists. According to this model, the language component's Mediator handles interactions with the user and with various subsystems that are highly competent in the performance of restricted and specialized tasks. In interacting with the user and the task specialists, the Mediator receives help from a user

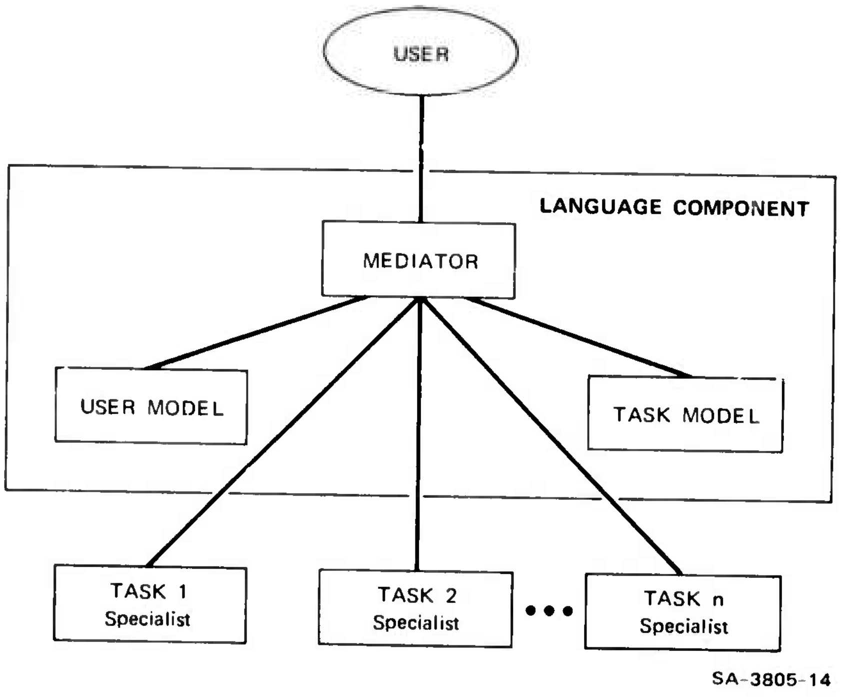


FIGURE 23 ABSTRACT LANGUAGE SYSTEM DESIGN

model and a task model. The user model encodes the discourse history, the state of the user's progress in the performance of some task, and information concerning other characteristics (e.g., acoustic parameters and, eventually, problem solving style) of the current user. The task model contains abstract descriptions of the task specialists' capabilities and of the tasks that the user himself can perform. Both the user model and the task model help the Mediator understand the user's utterances (principally by providing a context for the resolution of anaphora and ellipses). The task model helps the Mediator decide which task specialist to invoke in seeking to respond to a user's queries.

At the more concrete level of software modules, we are working with the system design of Figure 24 in mind. In this figure, double-lined arrows represent flow of control, solid arrows represent flow of information, and broken-line arrows represent subroutine calls. According to this system design, control is first resident in the Parser which receives data from the grammar and the lexicon (and, of course, from the user). In the course of its operations, the Parser is expected to call on the Composition Semantics Subprocessor, which builds up semantic interpretations of parse fragments. To perform the semantic interpretation, the Composition Semantics may request information from the Pragmatics/Discourse Unit or from the data base (through the Data Base Controller).

Once a semantic interpretation of a complete input has been constructed, it is passed to the Request Server. Since the CBC cannot carry out physical commands, user inputs are either requests for information or statements of fact for the system to commit to memory or use in updating its model of the task being performed. In either case, the Request Server calls the Data Base Controller, which may input new data into the data base (and indirectly cause updating-monitoring demons to be fired off), or may query the data base (and perhaps indirectly invoke knowledge specialists, such as the planner, that derive pieces of data not specifically encoded in the data base).

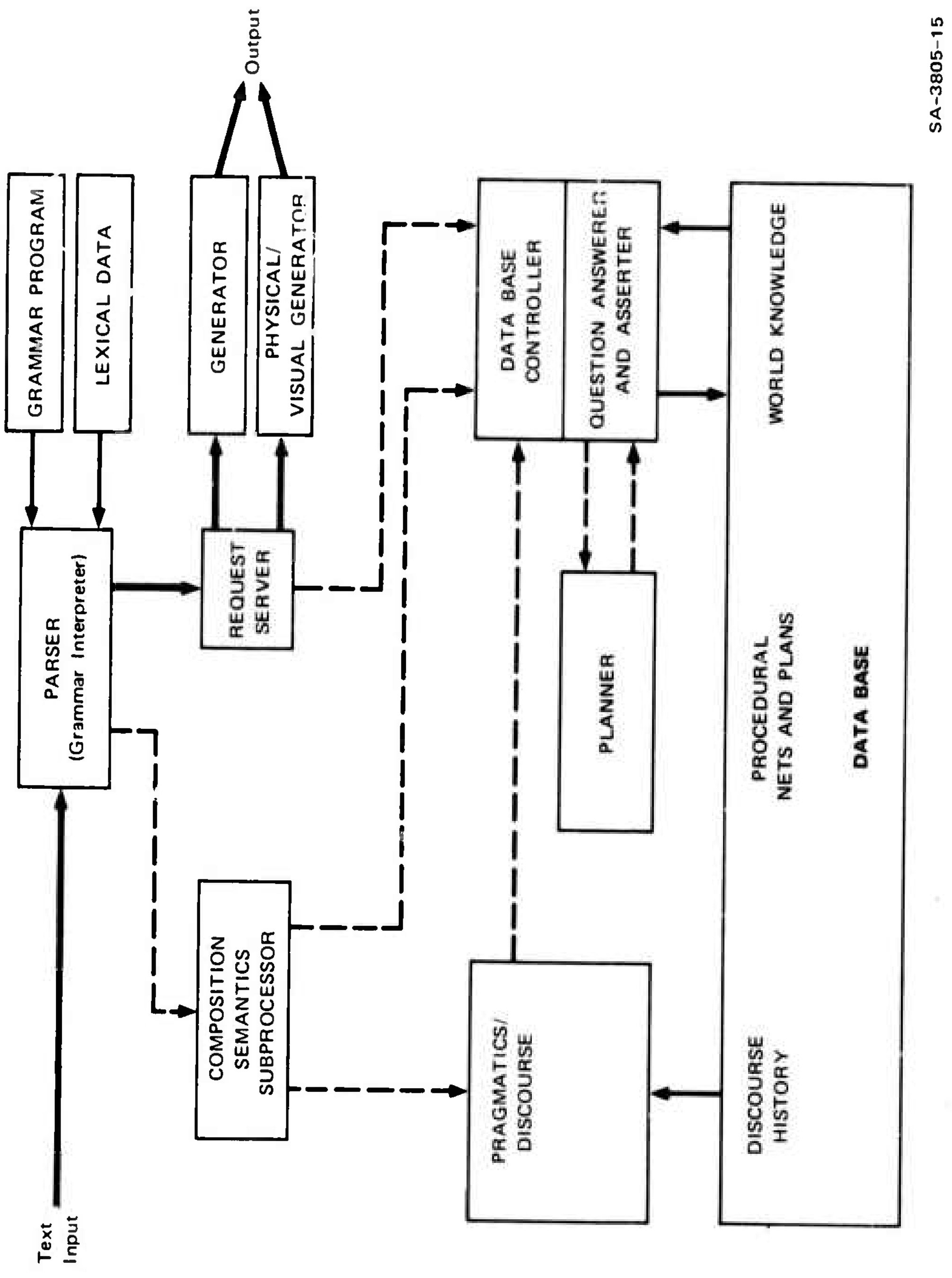


FIGURE 24 SOFTWARE MODULES

170

When an appropriate response has been determined by the Request Server, the response is passed either to the language generator (which produces spoken output), to the physical generator (which provides a mechanical output such as the pointing of the TV camera or the laser), or to both.

The relationship between the software-module system design and the more abstract design is indicated by Figure 25.

9. Semantic Representation

One of the most fundamental tasks in the construction of any artificial intelligence system is the selection and implementation of a scheme for representing knowledge. Within the past year, our research in semantics has concentrated on the development of a modeling scheme for use in natural language understanding that is capable of encoding multiple aspects of knowledge in a uniform, precise, and easily manipulatable form. Special emphasis has been given to the uniformity of representation since, depending upon the point of view of a processing algorithm or the disposition of the apprentice, a given piece of data may be regarded internally or expressed in output in a variety of ways.

The representation scheme that has been developed for use in the CBC is a variation of the semantic network schemes used by Simmons [39], Shapiro [40], Rumelhart and Norman [41], and other researchers. For natural language applications, networks have proven

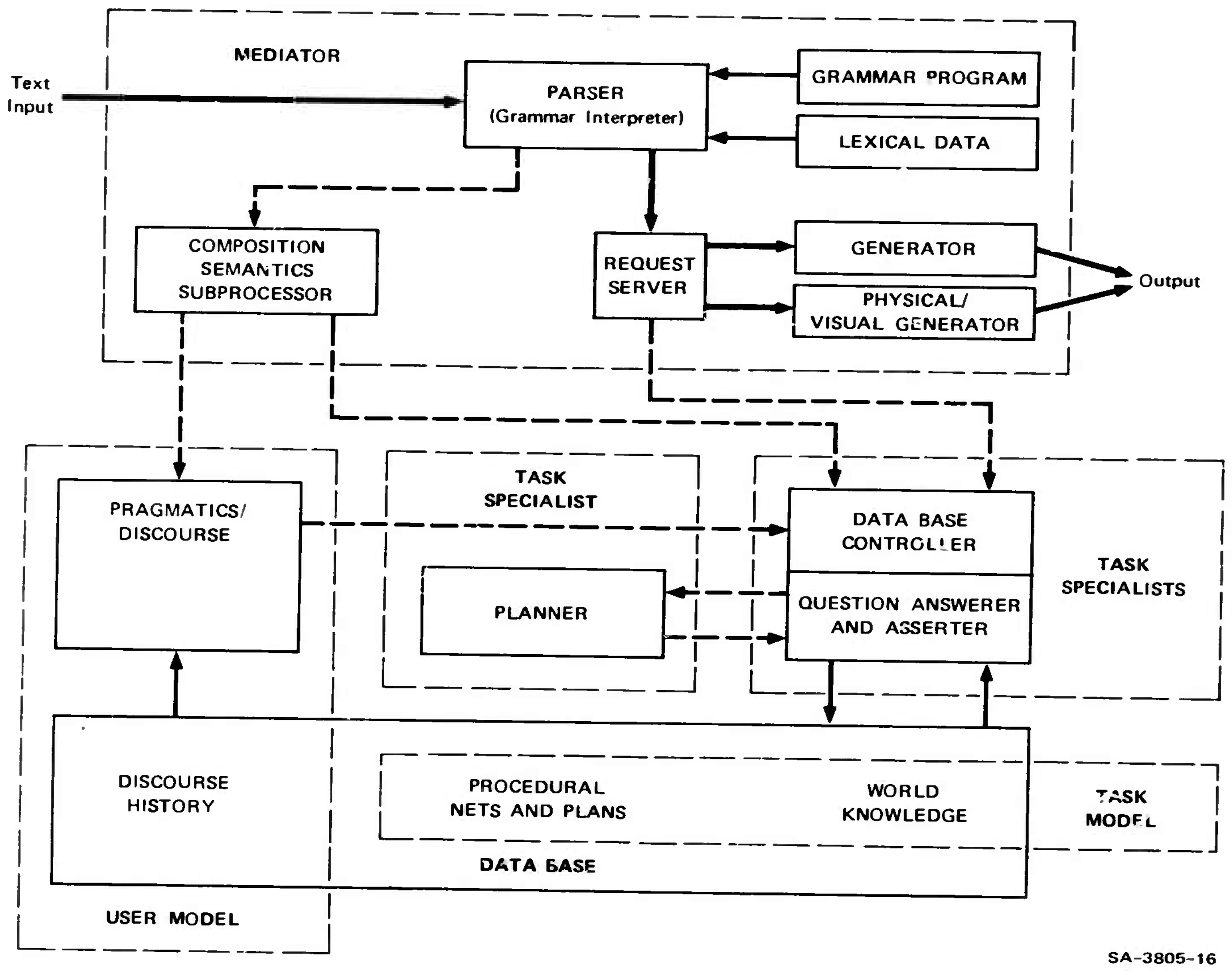


FIGURE 25 RELATIONSHIP OF ABSTRACT DESIGN TO SOFTWARE MODULES

to be very serviceable encoding structures, offering such recognized advantages as a convenient bidirectional linkage between semantically related data items and an inherent facility for associating deep conceptual case systems with event types. To extend the encoding power and flexibility of semantic networks, we have developed an augmentation that allows the nodes and arcs of networks to be partitioned into units called spaces. This allows knowledge to be bundled into units, thereby helping to condense and organize the data base. Specifically, partitioning allows quantification, abstraction, and categorization to be handled easily by networks. Partitioning also facilitates the encoding of process information and the translation from English into network representations. (A discussion of partitioned semantic networks is contained in Ref. 42.

a. Basic Network Notions

In its simplest form, a semantic network is a set of nodes interconnected by an accompanying set of arcs. A node may be used to represent an object, where an object may be virtually anything, including physical objects, relationships, sets, events, rules, and utterances. Arcs are used to represent certain "primitive" omnichronic (i.e., time invariant) relationships. (Such relationships may also be represented as nodes.)

A feeling for how nodes and arcs are organized to represent various facts may be gained by considering the network of Figure 26. In this network the node 'PHYSICAL.OBJECTS'

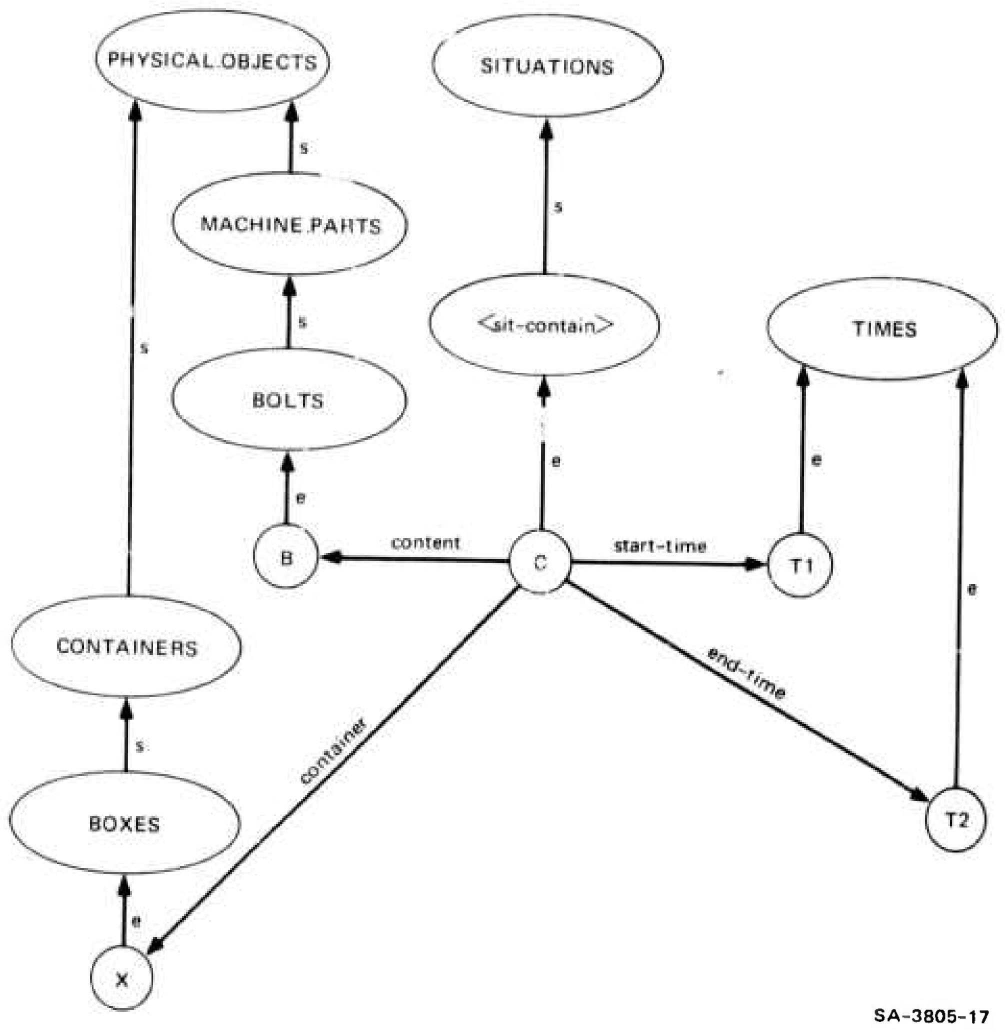


FIGURE 26 A TYPICAL NET FRAGMENT

(single quotes are used to delimit nodes) represents the set PHYSICAL.OBJECTS, the set of all physical objects. Likewise, node 'MACHINE.PARTS' represents the set of all machine parts. The arc labeled "s" from 'MACHINE.PARTS' to 'PHYSICAL.OBJECTS' indicates that MACHINE.PARTS is a subset of PHYSICAL.OBJECTS. Similarly, the network indicates that BOLTS is a subset of MACHINE.PARTS and that B, an Element of BOLTS, is a particular bolt. Following the hierarchy of another family, X is a particular box, an element of BOXES, a subset of CONTAINERS, a subset of PHYSICAL.OBJECTS.

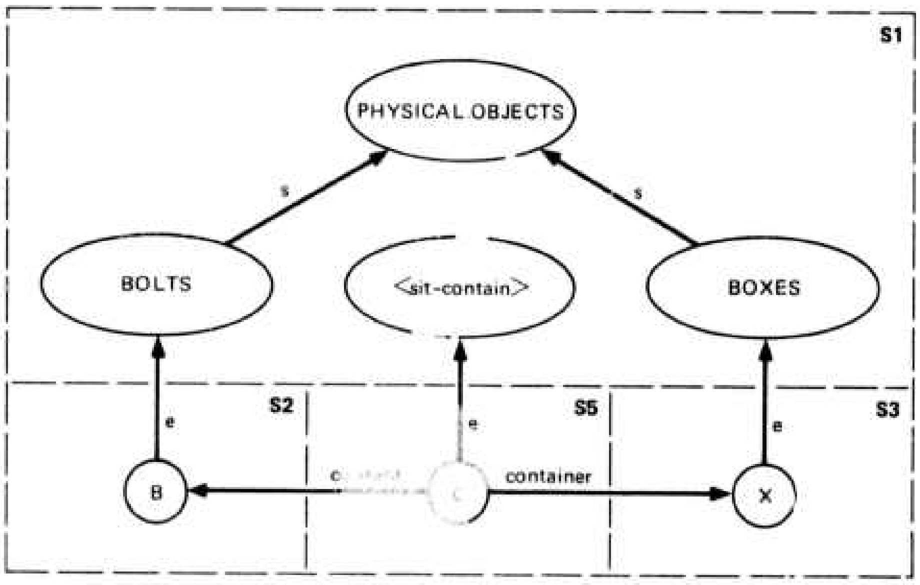
Node 'C' encodes a containing situation, an element of the situations set <sit-contain>, a subset of SITUATIONS, the set of all situations. In particular, 'C' represents the containing of bolt B by box X from time T1 until time T2. The various component parts of situation C are associated with it through special deep case relationships. For example, in the network there is an arc labeled "content" from 'C' to 'B'. This arc indicates that B is the #@content of situation C, where the notation "#@content of C" means "the value (#) of the content attribute (@) of C." Similarly, X is the #@container of C while T1 and T2 are the #@start-time and #@end-time respectively.

As a general principle, arcs encode only element, subset, and case relationships. (Under one interpretation, element and subset relations may be viewed as deep cases also.) Arcs are never allowed to encode relationships, such as ownership, which are time bounded.

b. Net Partitioning

The central idea of net partitioning is to separate the various nodes and arcs of a network into units called spaces. Every node and every arc of the overall network is assigned to exactly one space, with all nodes and arcs that lie in the same space being distinguishable from those of other spaces. While nodes and arcs of different spaces may be linked, the linkage must pass through certain boundaries that separate one net space from another.

Net spaces are typically used to delimit the scopes of quantified variables and to distinguish alternative hypotheses (during parsing and planning). However, before taking up such practical applications, consider the simpler (if atypical) network partitioning shown in Figure 27. Each space of the



SA-3805-18

FIGURE 27 A SAMPLE NET SPACE PARTITION

partitioning is enclosed within a dotted line. For example, space S1 is at the top of the figure and includes nodes 'PHYSICAL.OBJECTS', 'BOLTS', '<sit-contain>', and 'BOXES'. S1 also includes the two s arcs indicating that the set of BOLTS and the set of BOXES are subsets of the set of PHYSICAL.OBJECTS. In our diagramatic representations of semantic nets, an arc belongs to a space if, and only if, the arc's label is written within the dotted line boundaries of the space. Thus the e arc from 'B' to 'BOLTS' lies in space S2.

The various spaces of a partioning are organized into a partial ordering such as that shown in Figure 28. In viewing the semantic network from some point S in this ordering, only those nodes and arcs are visible that lie in S or in a space above S in the ordering. Thus, for example, from space S2 of Figures 27 and 28, only those nodes and arcs lying in S2 or S1 are visible. In

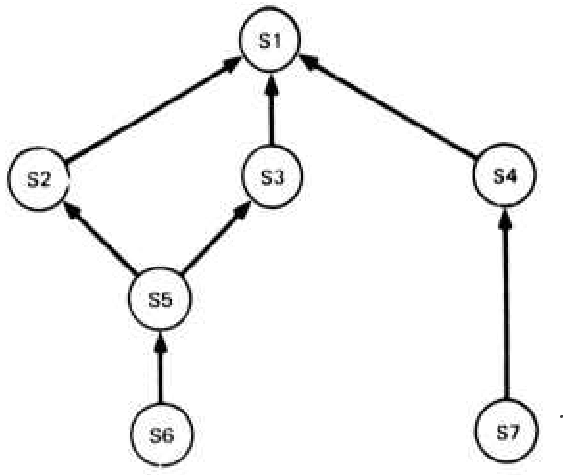


FIGURE 28 A NET-SPACE PARTIAL ORDERING

particular, it is possible to see that B is a BOLT and that BOLTS are PHYSICAL.OBJECTS, but it is not possible to see that X is a BOX. From space S5, information in spaces S5, S3, S2, and S1 is visible. Hence, from S5, the whole of the semantic network of Figure 27 may be seen. (For certain applications, the net may be inspected one space at a time. For example, it is possible to query the net in such a way that only nodes and arcs lying in space S2 are visible even though information in S1 is normally visible whenever S2 is inspected.)

In practice, partitioned networks are constructed by creating empty net spaces, adding them to the partition ordering, and then creating nodes and arcs within each newly created space. The use of partitioning in the encoding of quantified statements and categories is the subject of the next two sections.

c. Quantified Statements

In addition to an ability to encode specific facts (such as the containing event encoded in Figure 26), a semantic system needs some facility for grouping sets of similar facts into units--so as to allow those facts to be represented collectively through some sharing mechanism and to be conceptualized as an integrated whole. An ability to encode generalized information (in the form of quantified expressions) is of considerable importance since it is often impractical (or even impossible) to record the same information by a collection of individual specific statements, both because of the very number (possibly infinite) of statements required, and because details of particular individuals may not be explicitly known. Further, since quantification is a component of language, an ability to encode quantifiers is vital to the understanding of certain classes of English sentences. (Quantifiers were seldom used explicitly in the dialogs gathered in our protocol experiments concerning mechanical tasks. Nevertheless, quantification is needed for the representation of world knowledge.)

As an example of how quantification is handled in partitioned networks, consider the network of Figure 29 which encodes the statement

"Every bolt in the box is 3/4 inch long
and has a nut screwed onto it."

In this network, the node 'GS' represents the set of all general statements (the set of statements involving universal quantifiers or, under another interpretation, the set of recurring patterns of events). The node 'g' represents the particular statement (set of events) cited above.

Characteristically, a general statement encodes a collection of separate circumstances, all of which follow

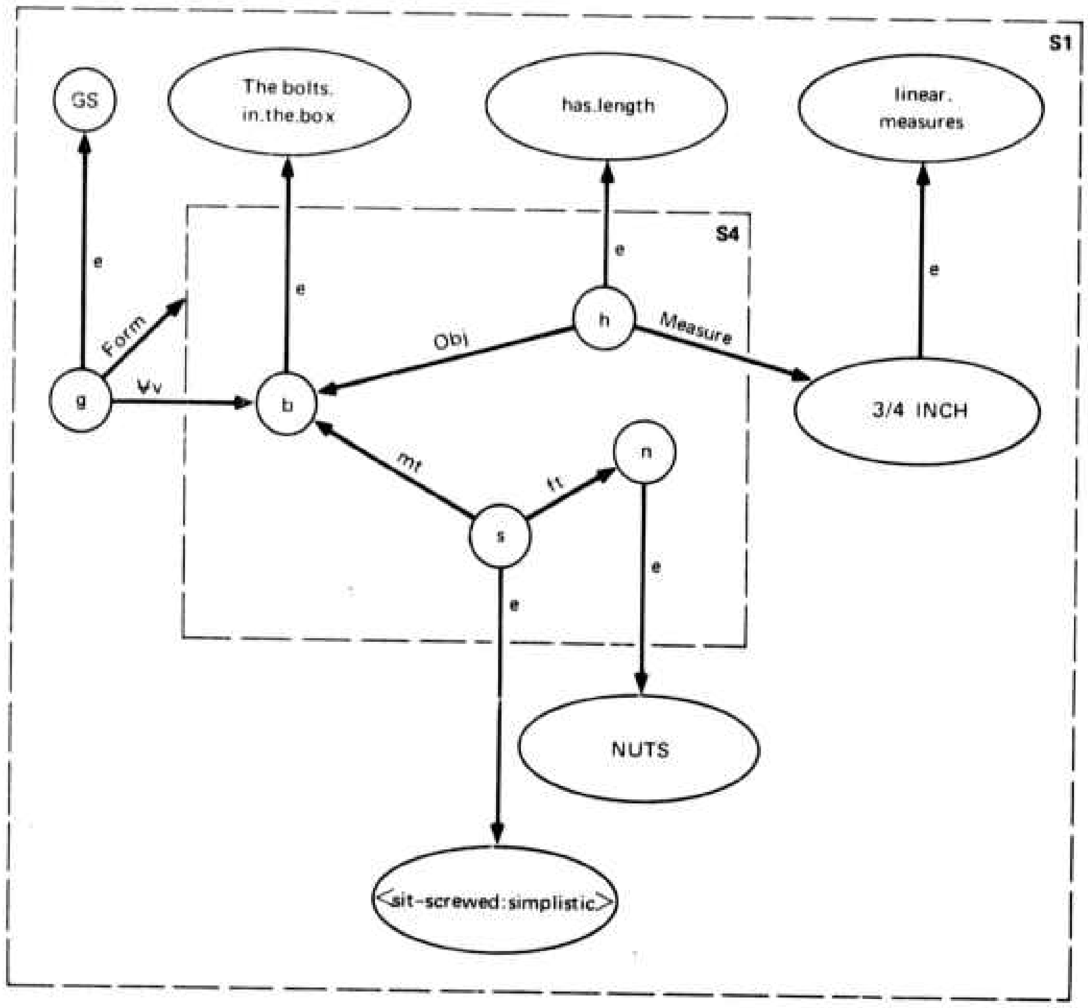


FIGURE 29 EVERY BOLT IN THE BOX IS 3/4-INCH LONG AND HAS A NUT SCREWED ONTO IT

the same basic pattern. This basic pattern is represented by the #@form of the general statement. The #@form of g is encoded by a net space, S4, which lies just below S1 in the partition ordering. (When one net space is pictured inside another, the inner space is below the outer in the partition ordering.) This net space may be thought of as a super-node, containing its own internal structure and representing a composite variable that takes on a different value for each of the instantiations of the recurring pattern. Each node and arc within the space of the super-node may be thought of as a subvariable.

General statements are also typically associated with one or more universally quantified variables, which are allowed to assume values in some specified range. Statement g, for example, has a universally quantified variable b given by its @Vv attribute. Note that variable b is necessarily a part of the #@form of g (i.e., 'b' lies in space S4). From node 'b' there is an e arc to the set the.bolts.in.the.box, indicating that the value of b (written #b) must be taken from the range set the.bolts.in.the.box. (The node 'the.bolts.in.the.box' has been created especially to help encode the general statement. Its meaning may be inferred subsequently when the.bolts.in.box.X is defined in paragraph "d" below.

The interpretation of a general statement is that for each assignment of the variables #@Vv to values in their corresponding ranges, there exist entities matching the structure of the #@form. For g this means that for every #b an element of the.bolts.in.the.box there exist

#h C <has.length>

#s C <sit-screwed;simplistic>

#n C NUTS

and the relations

#b is the #@obj of #h

3/4INCH is the #@measure of #h

#b is the #@mt of #s (i.e., #b is the male
-threaded part of #s)

and #n is the #@ft of #s.

Thus, the interpretation of q is that for every #b in the.bolts.in.the.box, there exists a situation #h in which the length of OBJect #b is the MEASURE 3/4 inch. Since '3/4INCH' lies outside space S4, there is only one measure for all the bolts in the box. Further, for each bolt #b there is a nut #n (depending on the individual #b) that is in a situation of being screwed onto #b. (A screwed:simplistic connection may exist only between two threaded objects, one with male threads (the #@mt) the other with female threads (the #@ft). A screwed:simplistic connection may be contrasted with screwed:standard connections in which multiple unthreaded parts are held together by a bolt (or other threaded object) that passes through the unthreaded objects to engage a nut.)

Complex quantifications entailing nested scopes may also be encoded by net spaces, as shown abstractly in Figure 30.

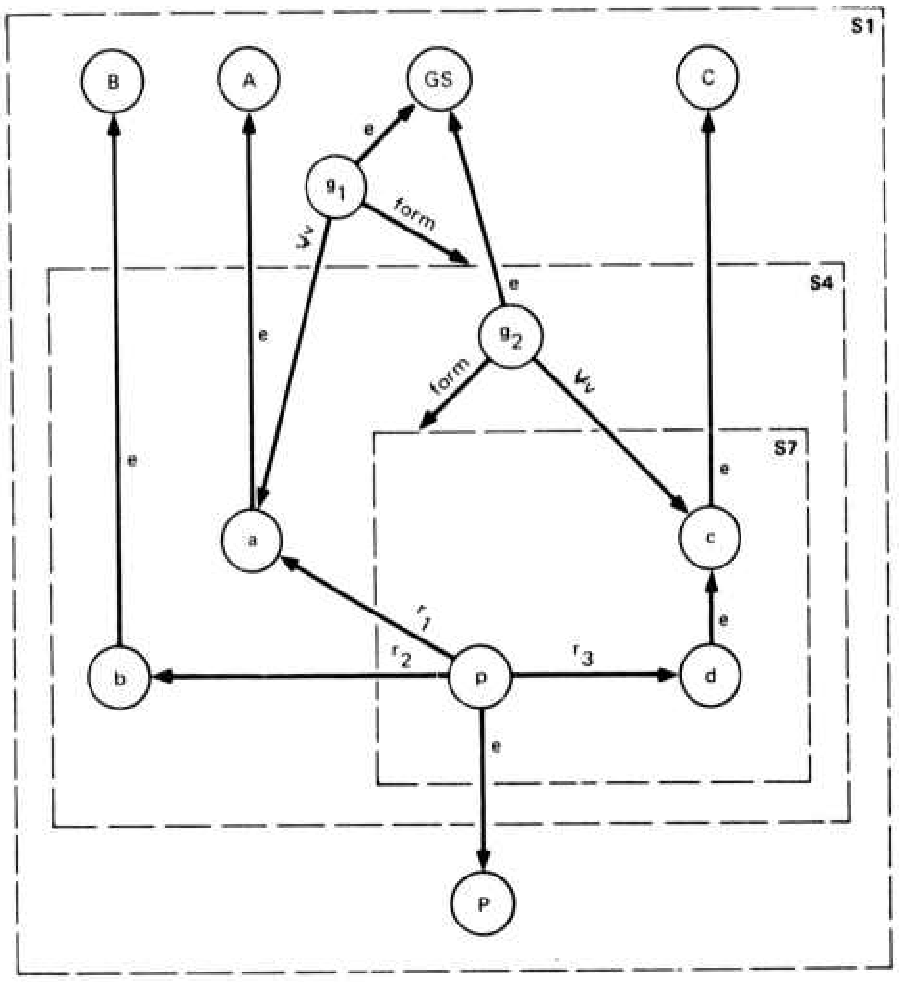


FIGURE 30 A COMPLEX ABSTRACT QUANTIFICATION
$(\forall a \epsilon A)(\exists b \epsilon B)(\forall c \epsilon C)(\exists d \epsilon c)[P(a,b,d)]$

d. Rules and Categories

A convenient method for organizing information in a semantic system is to divide the various objects (including physical objects, situations, and event objects) in the semantic domain into a number of categories. Using categories, objects that are somewhat alike become grouped together, allowing similar objects to be thought about and talked about collectively. The scheme is hierarchical in that some categories may be subcategories of more general classes. The lower a class is in the category hierarchy, the more alike its members must be. The "likeness" arises in that members of each category possess certain common characterizing properties (such as deep conceptual cases or an association with a common attribute.

The categorical system serves the important purposes of spotlighting similarities among objects and compressing redundant information by recording common information at the category level rather than with the individual. If an object Z is known to belong to some category K, then Z is known to possess the common properties of K's members and the common properties of the members of each of K's supercategories. This ability to encode information at the category level rather than with each individual is of practical importance because it saves computer memory and because all the elements of a category (perhaps being infinite in number) may not be explicitly known.

For natural language processing, the category system has the important feature that members of the more significant categories (the categories commonly held in the minds of humans) are expressed by the same set of linguistic patterns. As an elementary example, screwdrivers, wrenches, hammers, and saws belong to a category of objects that may be expressed by noun phrases headed by the noun "tool." Various attaching events may be expressed by complete sentences using the words "attach," "mount," or "fasten" as their central verbs.

Central to the notion of a category is the notion of a rule that specifies a necessary and sufficient test for category membership. Necessary rules, which all category members must obey, and sufficient rules, which can prove that an object belongs to a given category, are also of importance.

As a simple example of a category and its defining rule, consider the category of bolts in box X. This category is represented by the node 'the.bolts.in.box.X' of Figure 31, with the associated rule being encoded by net space S2. The

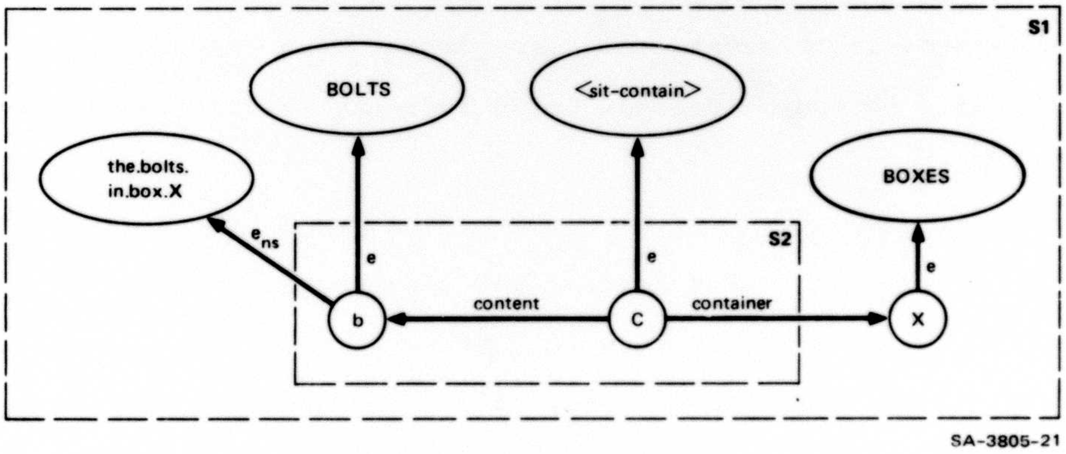


FIGURE 31 THE NECESSARY AND SUFFICIENT RULE DEFINING "THE BOLTS IN BOX X"

"ens" arc lying in space S2 from node 'b' to 'the.bolts.in.box.X' indicates that 'b' represents what may be thought of as an archetypal element of the category. (Symbol "ens" means architypal element, necessary and sufficient.) Any object with the characteristics of b belongs to the category, and all members of the category have the characteristics of b. As encoded in space S2, the characteristics of b include membership in BOLTS (the set of all bolts) and involvement as the #@content in a containing situation in which box X is the #@container.

In natural language processing, particularly during the parsing phase when surface structures are being translated into nets and when the semantic well formedness of sentences and sentence fragments is being tested, it is important to know what attributes (deep cases) are associated with certain categories of objects (especially with event, situation, and other verb-like categories) and what range of values each attribute may assume. This information has utility because attributes indicate the types of participants that are involved in particular categories of situations and because there is often a direct mapping from syntactic cases (including prepositional phrases) to these attributes. Knowing the correspondences between surface cases and attributes and knowing the ranges of values for each attribute allow some parses to be rejected on macrosemantic grounds and provide a facility for predicting the citing of certain situation participants in the surface language. (This prediction ability is especially important for speech understanding.)

ne attribute-range information for a category, collectively referred to as the category's delineation, may be associated with the category through a delineation rule. A delineation rule is a necessary rule that includes range information about every attribute of the delineated category.

As an example of a delineation rule, consider the delineation of category <to-bolt>, the category of events in which two machine parts are attached by using bolts as fasteners. Delineation information for this category is encoded by the network of Figure 32. In this network, the node '<to-bolt>' is linked to a node 'b' by an ed arc which indicates that b is the delineating "element" of <to-bolt>. Encoded within space S4 is attribute-range information concerning each of the six attributes possessed by members of <to-bolt>. In particular, the rule encoded by space S4 indicates that a bolting event involves an #@actor taken from the set of INTELLIGENT.ANIMATE.OBJECTS, a #@minor-p and a #@major-p taken from the set of MACHINE.PARTS, a set of #@fasteners taken from the set of BOLT/NUTS (a bolt/nut is a bolt and a nut which work together to form a single fastener), a #@tool taken from the set of TOOLS (which includes hands and fingers), and a #@time taken from the set of TIME.INTERVALS.

Given the two sentences

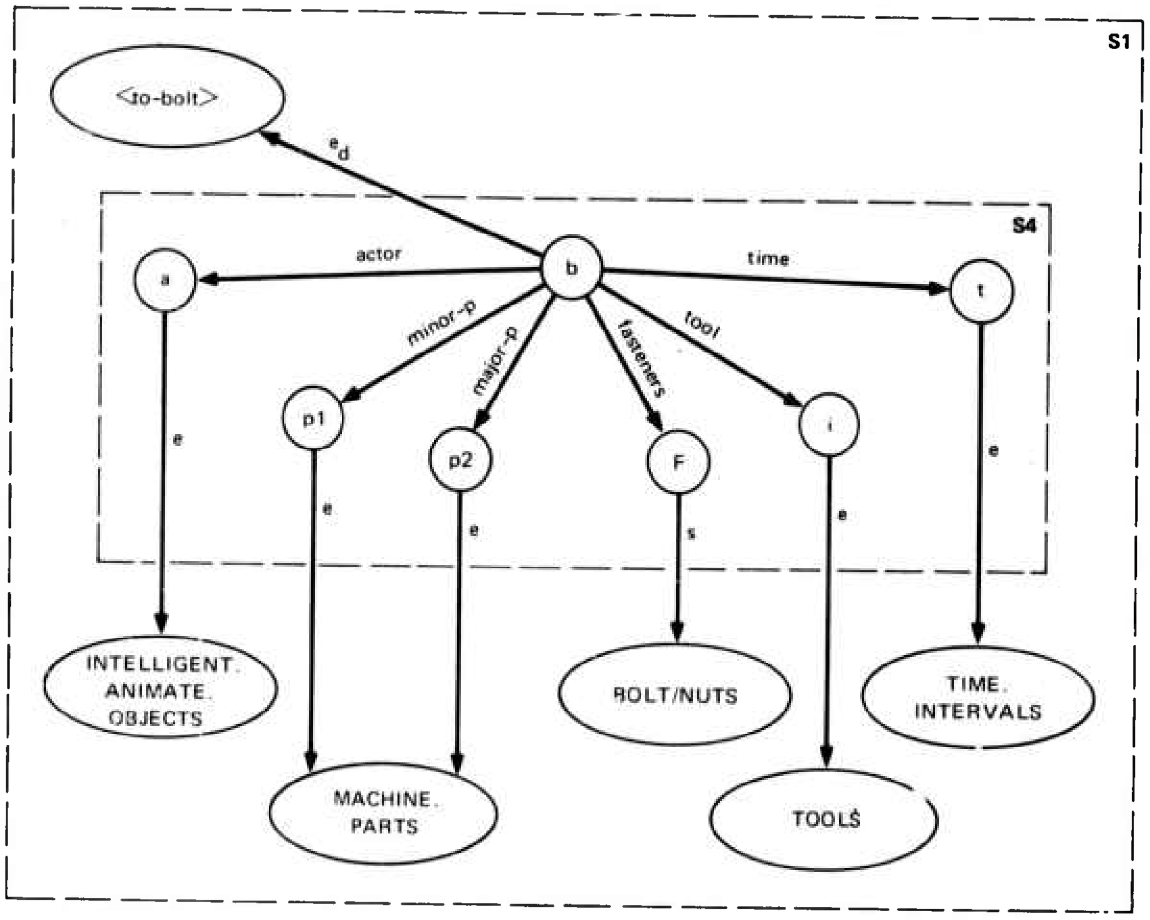
"I bolted the pump to the base plate WITH 1 INCH BOLTS," "ens" arc lying in space S2 from node 'b' to 'the.bolts.in.box.X' indicates that 'b' represents what may be thought of as an archetypal element of the category. (Symbol "ens" means architypal element, necessary and sufficient.) Any object with the characteristics of b belongs to the category, and all members of the category have the characteristics of b. As encoded in space S2, the characteristics of b include membership in BOLTS (the set of all bolts) and involvement as the #@content in a containing situation in which box X is the #@container.

In natural language processing, particularly during the parsing phase when surface structures are being translated into nets and when the semantic well formedness of sentences and sentence fragments is being tested, it is important to know what attributes (deep cases) are associated with certain categories of objects (especially with event, situation, and other verb-like categories) and what range of values each attribute may assume. This information has utility because attributes indicate the types of participants that are involved in particular categories of situations and because there is often a direct mapping from syntactic cases (including prepositional phrases) to these attributes. Knowing the correspondences between surface cases and attributes and knowing the ranges of values for each attribute allow some parses to be rejected on macrosemantic grounds and provide a facility for predicting the citing of certain situation participants in the surface language. (This prediction ability is especially important for speech understanding.)

"I bolted the pump to the base plate WITH THE WRENCH,"

the delineation of <to-bolt> may be used to determine that the WITH phrase in the first sentence supplies the #@fasteners case whereas in the second sentence it supplies the #@tools case.

The delineation rule of Figure 32 shows all delineation information concerning <to-bolt> to be encoded in a



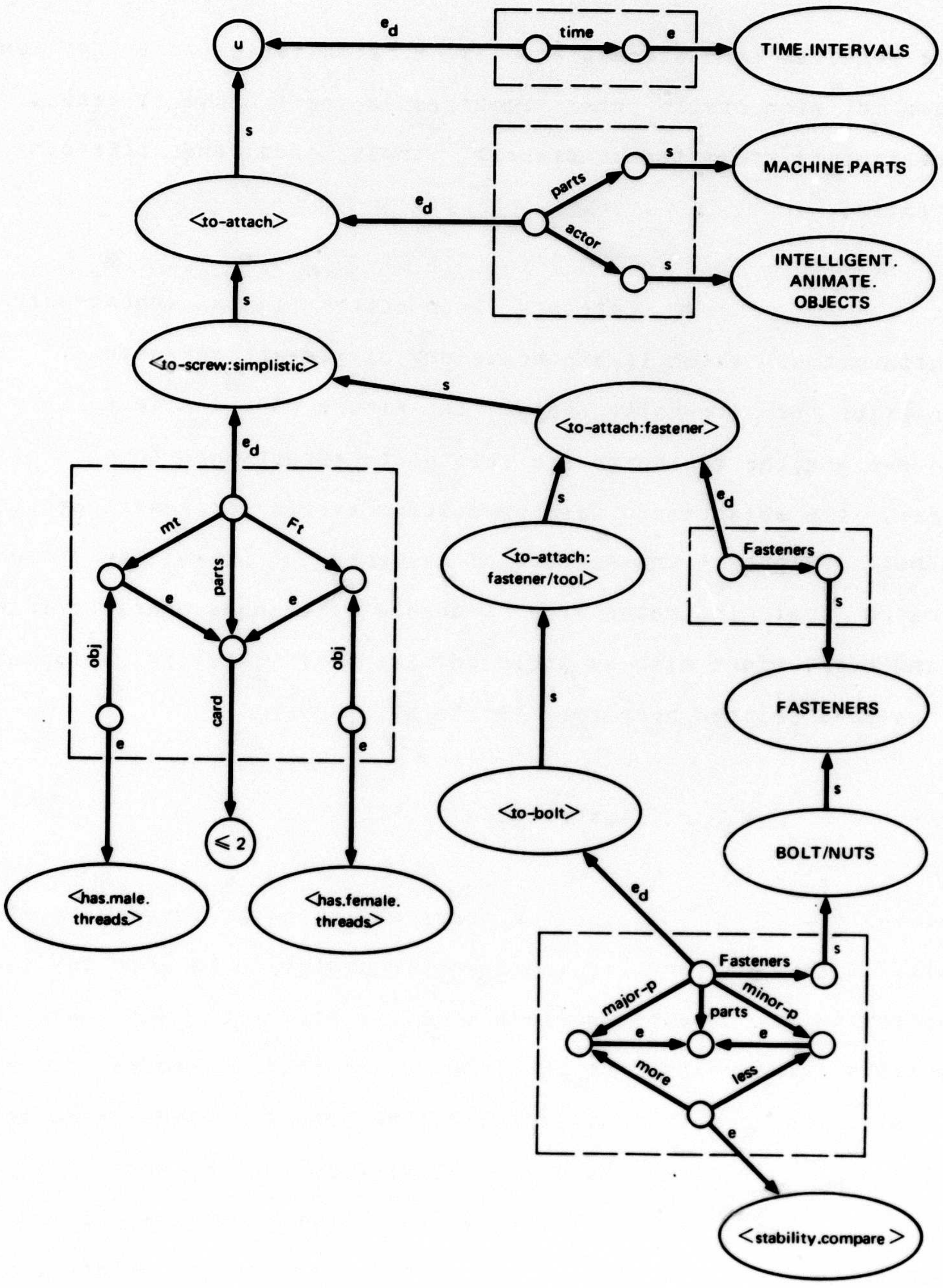
SA-3805-22

FIGURE 32 DELINEATION OF <TO-BOLT>

single rule linked directly to the category. In practice, categorical information is almost always distributed among many points in the categorical hierarchy. To see how information is distributed at various levels, consider the hierarchy of <to-attach> events that are shown in Figure 33. The most general category in the hierarchy is category U, the universal set. Even U has a delineation, since all objects (including events and situations) exist over some (possibly one-point or infinite) time interval. A subset of U is <to-attach>, the set of all attaching events of any nature whatever. Members of <to-attach> inherit the time attribute from supercategory U and add two additional attributes, #@parts and #@actor, of their own. In general, each attaching event involves a set of #@parts that an #@actor binds together in some way.

Two subcategories of <to-attach> are shown in Figure 33. The first is <to-screw:simplistic>, which is the set of events in which two threaded objects, one (#@mt) with male threads, the other (#@ft) with female threads, are engaged by twisting. Notice that the delineation rule of this category shows that the #@mt and the #@ft are both elements of the #@parts. The cardinality of #@parts is at most two (but could be one, as for a garden hose with one end attached to the other).

A second subcategory of <to-attach> is <to-attach:fastener>, the category of fastening events in which the #@parts are attached with fasteners. (Screwing a light bulb into a



SA-3805-23

FIGURE 33 THE <TO-ATTACH> FAMILY

socket requires no fasteners and is a simplistic screwing event. Nailing a sign to a post requires a nail as a fastener.) The delineation of <to-attach:fastener> simply adds the attribute of @fasteners.

Category <to-bolt> is a subcategory of <to-attach:tool> which is a subcategory of <to-attach:fastener>. The delineation of <to-bolt> shown in Figure 33 indicates how the #@major-p and the #@minor-p are related to #@parts and to each other. Further, the #@fasteners used by bolting events are restricted to be bolt/nuts as opposed to any type of fastener. Linkage to a process automaton that indicates the sequence of changes characterizing a bolting event might also be included with the category information but has been omitted here for simplicity.

e. Abstraction

Since a user may think at varying levels of detail, it is important for the semantic system to be able to encode information at multiple levels of abstraction and have some capability for jumping from one level to another. Figure 34 shows one way in which net partitioning may be used to encode an attaching event A at two levels of detail. By viewing the network from the vantage of space S2 (which lies below S1 in the ordering and is a sister space to S3), A is seen to be an element of <to-attach>, since the e arc lying in S2 is visible. Since the information lying in

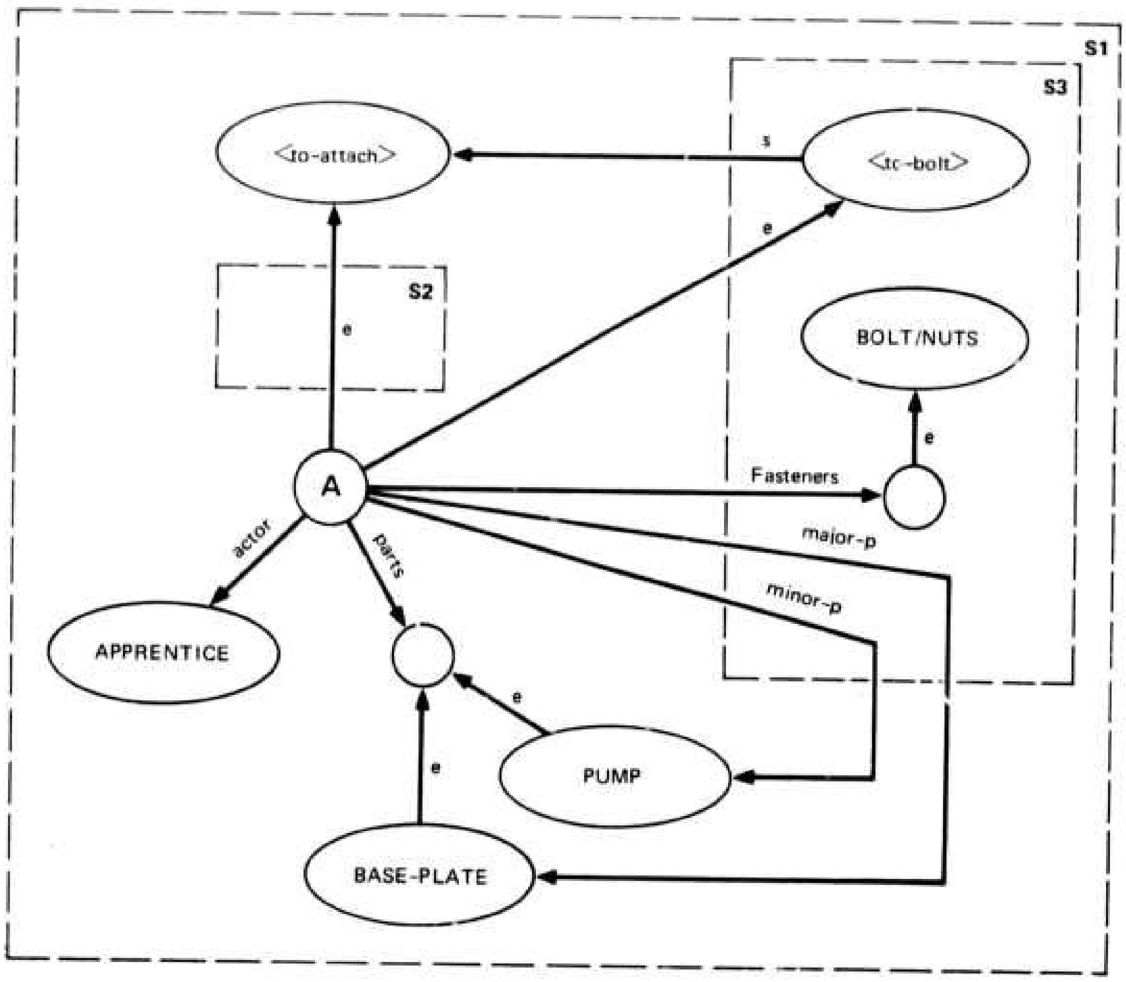


FIGURE 34 VIEWING A BOLTING AT TWO LEVELS OF DETAIL

S3 is invisible from S2, A appears to have only an #@actor and a set of #@parts and is not seen to involve #@fasteners. From S3 the same event may be viewed with more detail. First, the e arc from A to <to-attach> is invisible and A is thus seen as an element of <to-bolt>, a subset of <to-attach>. Further, at this finer level of detail, the #@fasteners involved in the attaching (bolting) event are visible (as are tools and the like, which are omitted from the figure for simplicity).

f. Processes

A very important aspect of the workstation domain is that of change. Since sequences of change tend to follow certain regular patterns, it is convenient to organize the recurring sequences of change into categories, grouping similar sequences together. Each category of sequential change is tantamount to an event category, the members of which may be thought of as individual enactments of a common plot or script which encodes a generalized pattern of change. For example, every event of tightening bolts follows the plot consisting of finding a wrench, putting the wrench on the bolt, twisting the bolt clockwise, and so on. Each enactment casts different actors in the various roles, but follows the same basic pattern.

Since the members of a particular event category may be distinguished as exactly those instantiations of sequential change that follow some particular script, the script itself forms the basis for a rule defining the event category.

During the past year we have been considering ways to encode process scripts in semantic networks for use in language processing. The procedural nets developed by the CbC planning group are, of course, a representation of process knowledge, and we anticipate the eventual merger of procedural and semantic networks. However, since procedural nets were not designed with language processing in mind, we have considered process automata [42] as a possible alternative. A process automaton is a section of semantic network that resembles a Mealy machine or a Woods AFSTN system [43]. While AFSTN was developed as a programming structure to describe the process of parsing language, the process automaton has been developed as a data structure for describing the processes (prototypal plots) cited by language and may be regarded as a parsing grammar that interprets (or generates) a sequence of changing conditions rather than a sequence of words. If a path can be found through a process automaton network for a given sequence of changes, then the sequence is accepted as a "word" (an enactment) in the "language" (category of events) defined by the "grammar" (process automaton).

During the next year, further study will be given to the problems of encoding scripts in networks; a common ground for procedural networks and process automata is expected to be found.

g. Translating into Semantic Networks

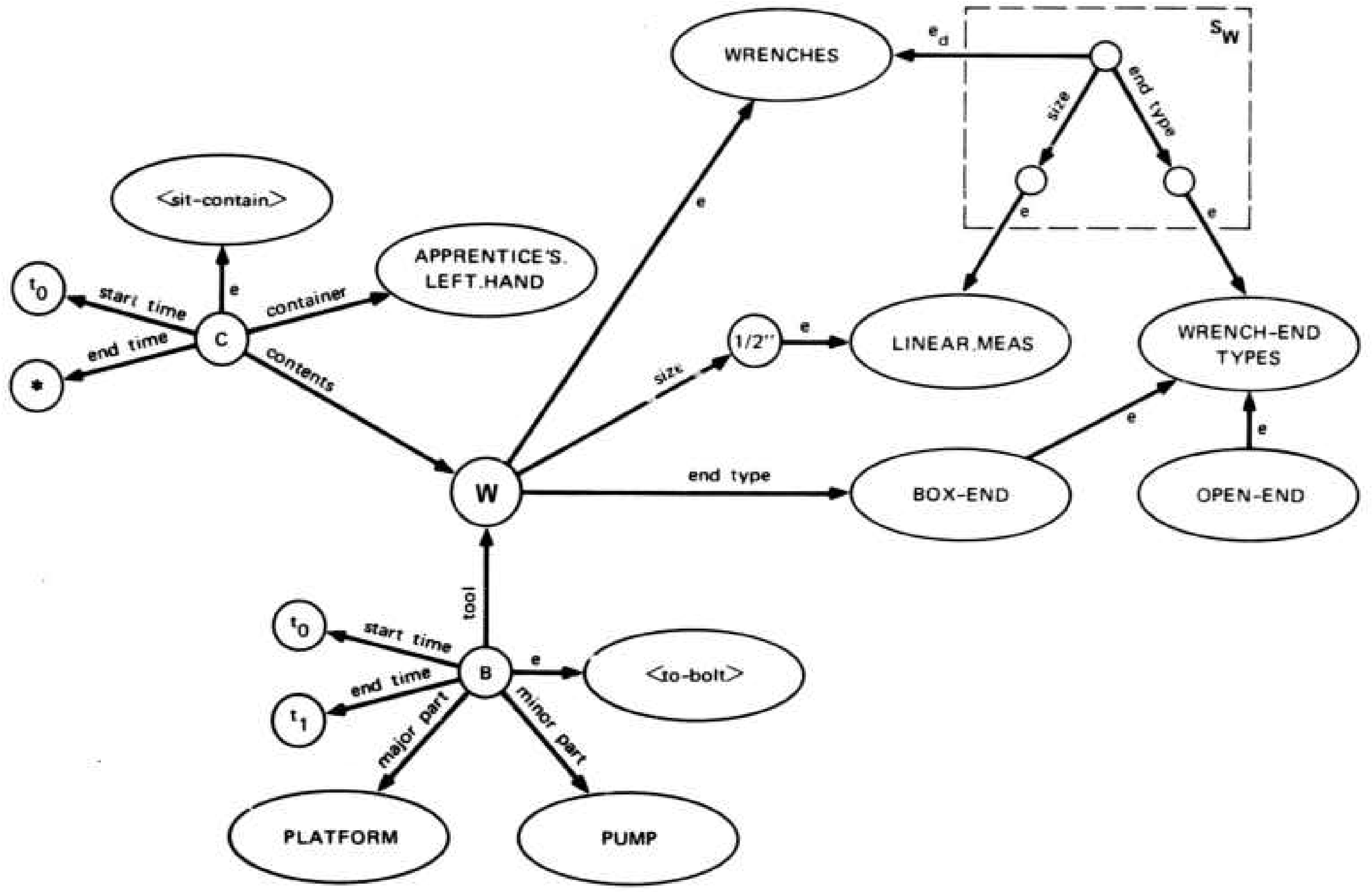
Although no attempt has yet been made to process natural language in the CBC domain, the semantic translation routines written for the SRI speech project were developed with the requirements of the CBC domain in mind. Since the techniques for building network representations of input sentences are discussed in

Ref. 36 the description will not be repeated here. With the recent modification of the speech parser, which allows it to accept text input, semantic routines for translating CBC related data are expected to be available within the next six months.

10. Relating the Semantic Structure to Discourse Needs

In the section on dialog analyses we mentioned two problems, reference and object description, which were closely related and which may now be discussed further. As mentioned earlier, solutions to both of these problems depend on being able to supply contextual focus, that is, an ability to restrict the system's attention to a small but pertinent subset of its total knowledge. Since the system's knowledge is recorded in a semantic network, a form of net partitioning may be used to group together those facts that are likely to be pertinent at a given point in the dialog. For task-oriented dialogs, the division of the dialog into cohesive subdialogs is closely tied to the task structure. This structure is embodied in the procedural net, which encodes the task structure in a hierarchy of subtasks. Grouping the information relevant to each subtask into a separate net space, and ordering the net spaces in accordance with the procedural net hierarchy, produce a knowledge structure that supplies contextual focus.

Figure 35 is the semantic net representation of a wrench W and its relationships to other objects (by "objects" we mean



SA-3805-25

FIGURE 35 SEMANTIC NET FOR WRENCH, W, WITH LOGIC PARTITIONING

any entity that is encoded as a node in the semantic net). Note that this is only a fragment of a semantic net: only a subset of the relationships in which W might participate are shown. The partitioning shown is the logical partitioning described in Section 8 above. Space Sw of this partitioning is used to delineate the class of wrenches, and indicates that each wrench has a #@size and an #@endtype. (These components of a wrench's description are indicated through case relationships because they are time invariant, intrinsic properties. Neither the size nor the endtype of a wrench may be altered without destroying the wrench itself. Node structures could have been used to encode this information, but such an encoding is more expensive and is not needed here.) Wrench W, an element of WRENCHES, has #@size 1/2-inch and #@endtype BOX-END. In addition to the intrinsic properties of size and endtype, wrench W has the distinction of having been used (as the #@tool) in the attaching of the pump to the platform between times Ti and Tj and of being in (being the #@content of) the apprentice's left hand from time Tk to the present. (NOTE: "Time" arcs go to intervals. An entity may also have a #@start-time and a #@end-time; in this case the interval is [#@endtime,#@starttime)).

All of this information is part of the history of wrench W. As such, any of it may be used in the description of W. However, in any given contextual focus only some of it is valuable. For this reason we would like to be able to highlight certain arcs and nodes in the network while they are in focus, letting them return to their unhighlighted state when the focus changes.

In order to do this, network partitioning is used in a new way. Nodes and arcs belong to both logical and focus spaces. The logical and focus partitions are orthogonal to one another in the sense that the logical space upon which a node or arc lies neither determines nor depends on the focus space in which the node or arc lies. (The focus partitioning differs from the logical one in several ways which we will discuss shortly).

The procedural net representation of a task encodes both the subtask hierarchy and the partial ordering of subtask performance for that task. For any given execution of the task, only a subset of the nodes in the procedural net are invoked. These correspond to the subtasks actually discussed by the apprentice and the expert. For example, if the expert directs the apprentice to attach the belt housing cover and the apprentice replies by saying that he has done that, then the nodes that correspond to details of how to perform the attaching are never invoked.

For each subtask entering the dialog a new focus space is created. The procedural net imposes a hierarchical ordering on these spaces. This hierarchy will be used, as the logical one is, to determine what nodes and arcs are visible from a given space. Note in particular that the arcs and nodes that belong to a space are the only ones immediately visible from that space. Arcs and nodes in spaces that are above a given space are also visible, but must be requested specially. Other arcs and nodes are not visible.

As mentioned previously, the focus partitioning differs from the logical partitioning in several ways. First, a node may appear in any number of focus spaces (but must appear in exactly one logical space). This happens when the same object is used in two different subtasks (e.g., the wrench of Figure 5). Either the same or different aspects of the object may be in focus in the two subtasks. It is also possible for a node or arc to be in no focus space. This just means that the object is not strongly associated with the performance of any particular subtask. For completeness, we define a top-most space, called the "communal space," and a bottom-most space, called the "vista space". The communal space contains those relationships that are time-invariant (e.g., the fact that tools are typically found in toolboxes) or common to all contexts. The vista space is below all other spaces and hence can see everything in the semantic net. This is useful for determining all of the relationships into which an object has entered.

Figure 36 shows the net of Figure 35 with a focus partitioning superimposed on the logical partioning. Focus Fi views wrench W as a box-end thath which is being used in the operation of bolting the pump to the platform. Focus Fj views the same wrench as one that is in the apprentice's left hand. The other information about the wrench (e.g., its size) is recorded in the communal space. All of the information is visible from the vista space.

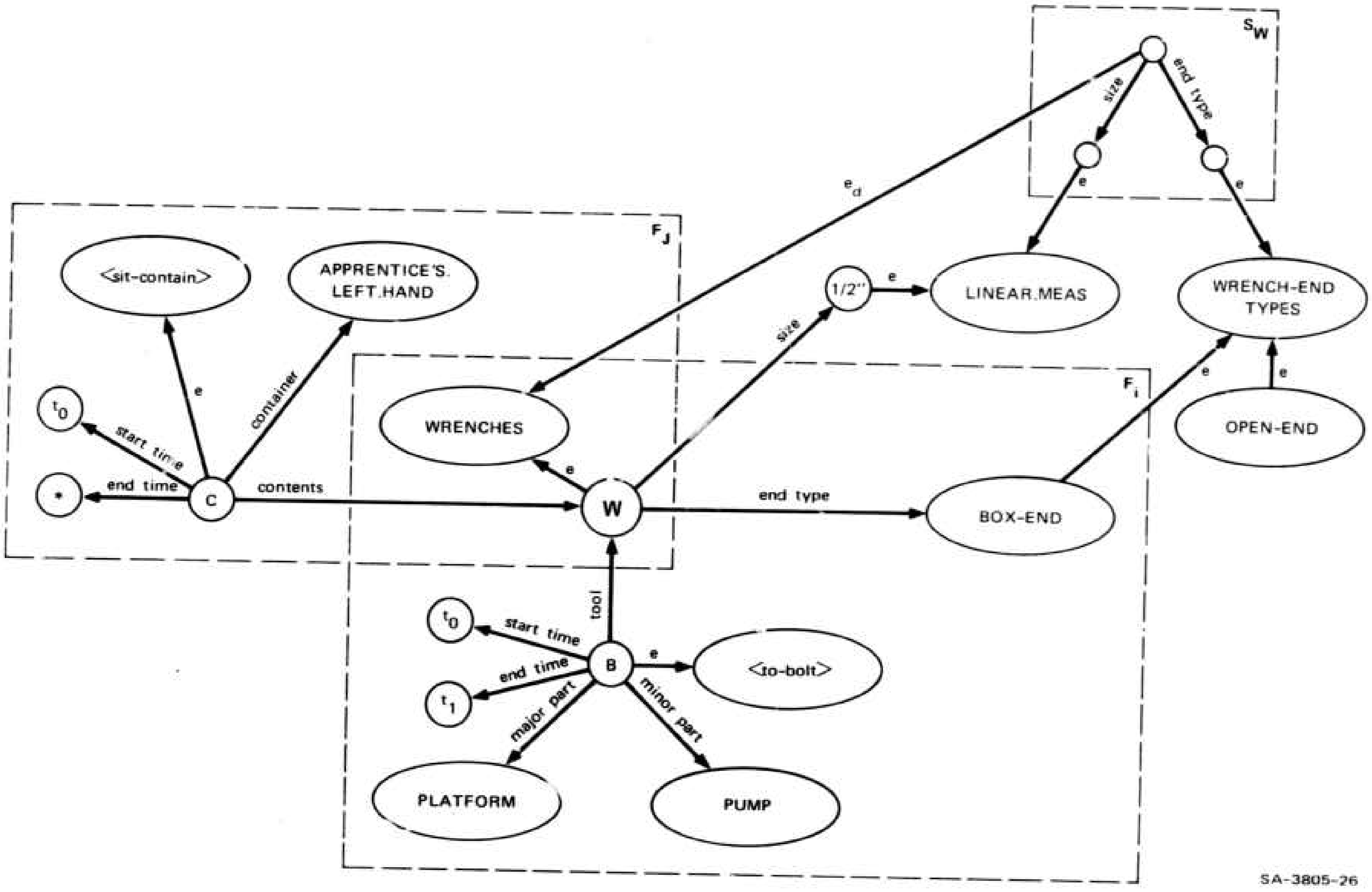
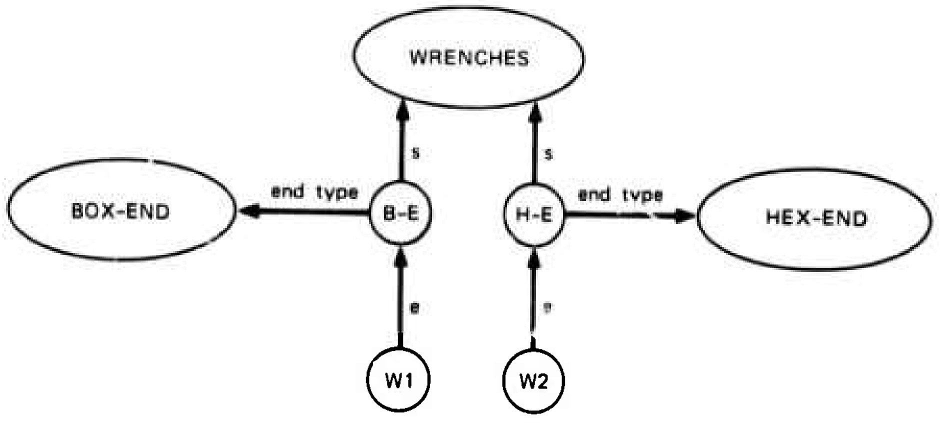


FIGURE 36 SEMANTIC NET FOR WRENCH. W. WITH FOCUS SPACE AND LOGIC PARTITIONINGS

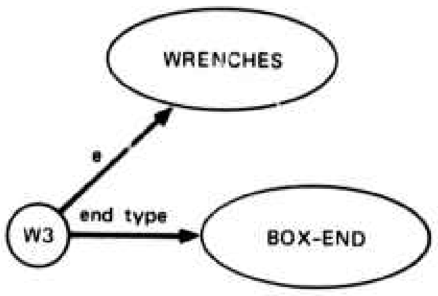
The representation of an object in a focus space will include only the relationships that have been mentioned in the dialog concerning the corresponding subtask or that are inherent in the procedural net description of the local task. The distinction between the verbal context and the general world knowledge context that we mentioned in Section 4 above (Dialog Analyses) may now be seen. The verbal context is supplied by the information recorded in the subspace hierarchy. The general world knowledge context is information that is present in the communal space. When resolving a reference, we can decide how to divide effort between examining links in the local space and looking back into the communal space.

An advantage of adding this new partitioning is that special information can be recorded at the local focus level. Thus if several links in the net must be followed to establish some fact about an object (i.e., some logical deduction must be done), the result of that work may be stored explicitly in the local focus space. The logical deduction does not have to be redone for local references. If this information is put in its own logic space, then it remains invisible from the knowledge net (the topmost logic space). For example, consider the situation portrayed in Figure 37. All of the nodes and arcs in this figure are in one focus space, F1. B-E is a set of box-end wrenches to which W1 belongs. H-E is a set of hex-end wrenches to which W2 belongs. If the apprentice now says, "... the box-end wrench", he means W1. The utterance level structure (created by parsing) for the phrase "the box-end wrench" is shown in Figure 38, and some amount of work must be done to establish the



SA-3805-27

FIGURE 37 SEMANTIC NET SHOWING MEMBERS OF TWO SUBSETS OF THE SET "WRENCHES"



SA-3805-28

FIGURE 38 SEMANTIC NET FROM PARSE FOR "BOX-END WRENCH"

correspondence between W1 and W3. However, it is quite likely that W1 will again be referred to as "the box-end wrench". By explicitly storing the box-end property of W1 in focus space F1, redundant work may be avoided. Figure 39 illustrates the new structure. Note that the e arc to WRENCHES and the end-type arc to BOX-END are in a separate logic space, L1. This makes them invisible at the knowledge net level. In fact, they are not visible from any logic partition outside of this focus space. This means the arcs essentially exist only for the duration of the subtask. In fact, once the subtask is completed, the special logic space may be deleted.

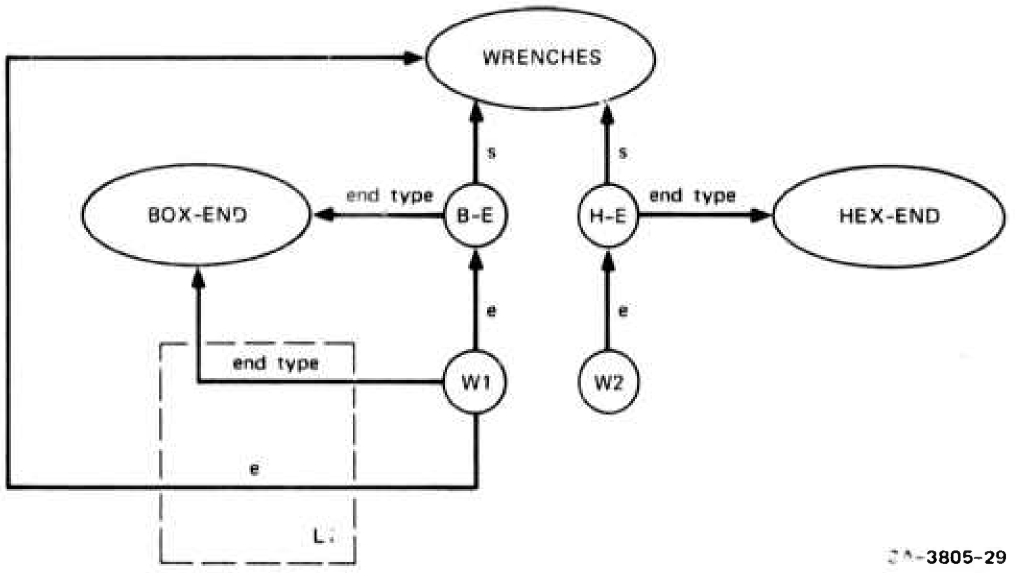


FIGURE 39 SEMANTIC NET SHOWING LOCAL FOCUS INFORMATION FOR BOX-END WRENCH, W1

The use of net space partitioning in several different dimensions is a powerful tool. We have only begun to investigate some of the possible uses and their interactions.

C. Problem Solving

1. Freedom of Movement (FM)

One way to increase the power of our models of mechanical equipment is to include in the models a representation of the freedom of movement existing between components of a device. Knowledge about freedom of movement enables the system to answer the following kinds of queries:

- What linear freedom of movement exists between two specified components in a specified configuration of the device? (E.g., "They are rigidly attached", "Part1 can move only in the +Y direction with respect to Part2", "Part1 and Part2 do not physically constrain each other", and so forth.)

- If Part1 is pushed in direction +X, which other components will move along with Part1, and which will remain stationary?

- How is Part1 supported?

and, by extension,

- If a specified operation is performed on the device (e.g., "Unscrew bolt B", "Slide the crankshaft in direction -X", "Turn the crankcase upside-down") will any parts lose their support, and hence fall out of place?

The last two queries require the ability to compute physical (as opposed to logical) support. Although there are other, more sophisticated ways to compute support, the "freedom of movement" approach to be described below often provides correct answers with relatively little computational cost.

Knowledge about freedom of movement is being incorporated into our models at a low level. Besides facilitating the computation of physical support, this knowledge can be used in the simulation of assembly-disassembly actions and in the automatic generation of disassembly subgoals. In the remainder of this section we will discuss our representation and use of freedom of movement in more detail.

a. Representing FM

Recall that a CONNECTION is defined for each pair of components that are in contact in the canonical model of the fully assembled device. We characterize the freedom of movement (henceforth abbreviated "FM") restrictions that the two components of the CONNECTION impose on each other when they are POSITIONED or ATTACHED to one another, and associate this information with the CONNECTION. The FM restrictions imposed by a CONNECTION could be represented in several different ways; for example:

- As a list of restricted directions and/or rotations

- As a set of linear constraints relating the positions of the two parts

- Deduced directly from a geometric representation of the shapes and positions of the parts (say, by displacing one part and seeing if it intersects the other).

Other representations are possible, including hybrids of two or more of these.

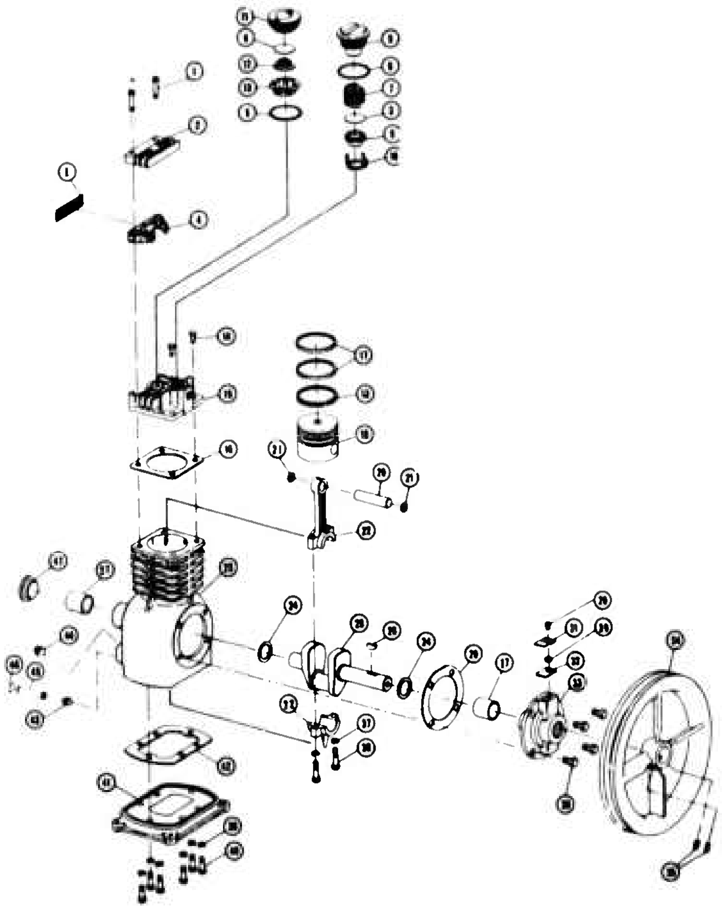
We have implemented an FM model of the air compressor pump, using a simplified version of the first of the above representations, in which only translational restrictions and not rotations are taken into account. The current version of this model makes the simplifying assumption that only six directions exist, corresponding to a standard set of three-dimensional axes and denoted +X, -X, +Y, -Y, +Z, and -Z. In this model, CONNECTIONS are classified into Types, where the type characterizes the shape of the contact boundary between the two parts. The CONNECTION types represented are:

> ABUTS -- implies a flat, surface contact. Each part restricts the FM of the other in one direction.
>
> SPINDLES -- a shaft of one part goes through a hole in the other. FM is restricted in every direction except along the axis of the shaft.
>
> MESHES -- one part is slid or seated into an indentation in the other. FM is restricted in all but one direction.
>
> SCREWS-IN -- One part screws into the other. FM restrictions depend on the state of the screw CONNECTION (tight, loose, just positioned, or removed).

This information is stored with the CONNECTION as the value of the QLISP property "TYPE." These types were chosen for their intuitive appeal, based on kinds of connections that actually occur in the pump. Of course, any number of additional types can be defined, but we would like to get by with as few as possible in order to keep the model simple and general. We have found the above four types, used singly or in combination, sufficient to represent the connections occurring in the pump--a reasonably complicated piece of equipment, as indicated in Figure 40.

With each CONNECTION is stored information pertaining to the orientation of the contacting surfaces. For example, suppose three abutting parts A, B, and C are lined up along the X axis as in Figure 41. We have two CONNECTIONS of type ABUTS--(A,B) and (B,C). The directional information is stored symmetrically so that if we are interested in moving part A, we discover that it is restricted by part B in direction +X; whereas if we want to move part B, we find that it is restricted by part A in direction -X (similarly for the CONNECTION between B and C. This information is stored redundantly on the CONNECTION's property list under the indicator DIRECTIONS; for example, (... DIRECTIONS (A +X B -X) ...). As long as movement is restricted to translations, FM restrictions are transitive and can be viewed as propagating through any rigid part. Now suppose that we push part A in direction +X. We find that part B restricts part A's FM in direction +X; likewise, part C restricts part B's FM, also in direction +X. In effect, the restriction "propagates" from part C through part B to part A, so that part C also restricts part A in +X.

Of course, this simple an approach to representing movement restrictions will not always work perfectly. For example, suppose parts A, B, and C are arranged as in Figure 42.



Reference Number	Part Number	NAME OF PART	Quan. Used	Reference Number	Part Number	NAME OF PART	Quan. Use
1	96472	Head Bolt	2	30	21614	Breather Valve Spacer	1
2	45007	Cylinder Head Cover	1	31	38941	Breather Valve Plate	1
3	45020	Filter Screen	1	32	38944	Breather Valve	1
4	45021	Filter	1	33	45003	End Cover Assembly-includes Ref. No. 27	1
5	45024	Outlet Valve Stop	1	34**	45025	Pulley for 101 TV	1
6	45014	Copper Washer	2	35	98627	Pulley Set Screw-5/16-18 x 1/2" Socket Head	2
7	45011	Outlet Valve Spring	1	36	18060	End Cover Bolt-1/4-20 x 1/2"	4
8	45013	Valve	2	37	9522	Lockwasher-1/4"	2
9	45016	Outlet Valve Seat	1	38	11109	Hex Head Bolts-1/4-20 x 1"	2
10	45019	Outlet Valve Retainer	1	39	9522	Lockwashers-1/4"	6
11	45017	Inlet Valve Seat	1	40	980	Hex Head Bolts-1/4-20 x 3/4" long	6
12	45012	Inlet Valve Spring	1	41	45006	Base Plate	1
13	45018	Inlet Valve Retainer	1	42	45009	Base Plate Gasket	1
14	18061	Head Bolt	2	43	9349	Pipe Plug 1/8"-Furnished with 102,17500	
15	45005	Cylinder Head	1			only	1
16	45008	Cylinder Head Gasket	1	44	9348	Pipe Plug 1/4"	1
17	31901	Piston Rings	2	45	9609	Pipe Nipple-1/8" x 5"-Furnished with com-	
18	17083	Oil Ring	1			pressor only	1
19	38932	Piston	1	46	14292	Pipe Cap-1/8" Furnished with compressor	1
20	12992	Wrist Pin	1			only	
21	7890	Wrist Pin Retainer Washer	2	47	45027	Crankcase Plug	
22	45001	Connecting Rod Assembly	1	Not Shown	43797	Parts List	.
23	45004	Crankcase Assembly-includes Ref. No. 27	1	**	45050	Pulley for 100 TV	1
24	45023	Thrust Washer	2	*	45051	Crankshaft for 100 TV	1
25*	45002	Crankshaft for 101 TV	1	Not Shown	45060	Head Plate Spacer	2
26	9397	Woodruff Key No. 9	1	Not Shown	45244	Set-Gaskets	1
27	45026	Bronze Bushing	1	Not Shown	45237	Set-Rings	1
28	45010	End Cover Gasket	1	Not Shown	45245	Set-Valve & Spring	1
29	38951	Breather Valve Screw-8-32 x 5/16"	1				

SA-1530-57

FIGURE 40 PARTS LIST FOR THE PUMP UNIT

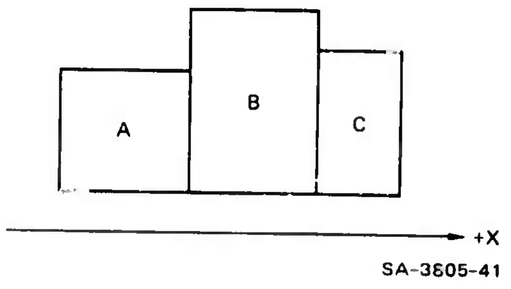


FIGURE 41 ABUTTING PARTS

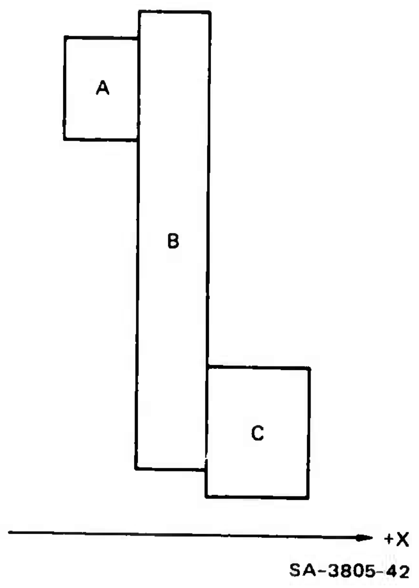


FIGURE 42 ABUTTING PARTS NOT SQUARELY ALIGNED

Here C does not really restrict A's movement in the +X direction. Many parts of the compressor are lined up squarely one against another, so that this kind of problem often does not arise and the simple model suffices. We intend to add to the model some notion of shape and position so that such cases can be recognized and handled correctly. The semantics of assembly/disassembly problems suggests that parts are usually held or fastened in specific "canonical" configurations, and subassemblies are set down in stable positions during the assembly/disassembly process. Thus it might be argued that the model need not represent "jumbled" configurations such as would result from pushing Part A to the right in Figure 42. It would, however, be desirable for the system to recognize when such "jumbled" configurations might occur, and warn the apprentice accordingly.

Besides TYPE and DIRECTIONS, some CONNECTIONS have the OLISP property STICKY. A true value of this property indicates that there is enough lateral friction in the connection to appreciably restrict freedom of movement.

b. Using FM in Modeling and Planning

Saying that "part B restricts part A's freedom of movement in direction +X" is equivalent to saying "if you push part A in direction +X you are also pushing part B in the same direction". Ignoring for the moment the effects of gravity and friction, we can model the effects of pushing part A in direction +X as follows.

We find all the parts that immediately restrict A's freedom of movement in direction +X; in other words, parts that have CONNECTIONS with A that impose the given restriction on A's FM. Then, for each such part we find any other parts that immediately restrict its FM in the +X direction; we continue finding all the parts so influenced (keeping track of which parts we have already processed so that we do not get caught in an infinite loop). The final list of parts contains all those parts that also get pushed in the +X direction when we push on part A.

For each part P found by the above algorithm, the system remembers the path of CONNECTIONs originating from part A and leading to the given part P. This path represents a kind of cause and effect chain that "explains" why part P has to move when we move part A. If any link in this chain can be broken (by altering one of the individual CONNECTIONs in such a way that the given FM restriction is no longer imposed by that CONNECTION -- e.g., by removing a fastener), then we may be able to break the chain of causality itself and hold part P (and all parts further down the chain) stationary while we move part A (and all the parts between A and P). This type of analysis can generate subgoals for the removal and installation of parts without having to store such subgoals explicitly. For example, suppose the main goal to be achieved is to remove the crankshaft from the pump. In order to remove the crankshaft, it must of course have freedom of movement in some direction with respect to the rest of the pump. We may query the model to see if such freedom of movement already exists. If not, the model can return a summary of which chains of parts and CONNECTIONs are responsible for restricting the crankshaft's movement. We can then propose as subgoals the removal of these restrictions, usually by unfastening and/or removing intermediate parts. Plans generated in this fashion may entail moving whole "chunks" of parts at once, since the planner first achieves freedom of movement for some part and then simply moves the part in the desired direction along with any other parts that may be pushed along with it.

c. The MOVE Function

The function (MOVE part1 dir part2list) computes the effects of moving part part1 in direction dir while holding the parts on part2list stationary. The actual workings of this function are somewhat more complicated than is indicated above. It takes into account that if one part supports another, then the supported part will tend to follow along when the supporting part is moved in any direction, unless the supported part is explicitly held back. (In the latter case the part may become unsupported, a condition that the system will detect.) Also, some restrictions on FM are softer than others, in the sense that they can be overcome by applying sufficient force to the parts involved. An example of a soft restriction is a force fit, or one part sticking to another.

Suppose that the +X direction in the above example corresponds to "up" in the real world (in other words, we introduce a force of gravity into our model that pulls all parts in the -X direction). Then the chains of FM restrictions discussed above may indicate which parts are supported by part A. It is true that "restricts movement in the up direction" is not always synonymous with "is supported by" since the "supporting" part must in some cases be centered under the other part to truly support it. The current model does not include any knowledge about whether parts are centered in this way. At some cost in additional computational complexity, we can add such information in symbolic form, or perhaps go as far as deriving it from a three-dimensional model of shapes and forces [44].

The current model recognizes (but makes no attempt to analyze) the case of an object supported jointly by several others. However, in the case of a single chain of FM restrictions leading from a given object to a "support object" (such as a table or the apprentice's hand) we can treat this chain as the supporting chain for the object.

Essentially, the MOVE algorithm works by "probing" one or more parts in a given direction and tracing through the chains of hard and soft FM restrictions. By probing the part that is to be moved in the direction indicated, the system finds other parts that must move as well. By probing the parts that are to remain stationary in the opposite direction, it finds parts that must remain stationary. Then these sets of parts are probed upwards to find probable chains of support, and warn of possible loss of support that may result from the move.

If the system finds that the desired move is impossible due to some chain of hard FM restrictions, the following procedure is followed.

Each local FM restriction making up this chain is classified according to the local precondition(s) that must be achieved in order to "undo" the restriction. The particular preconditions that apply are determined from the type of CONNECTION, the fastening if any, and the direction in which the parts are being probed. Table 5 summarizes these preconditions in the estimated order of increasing difficulty to achieve them.

When two parts are found to have an FM restriction preventing a desired move, local subgoals for breaking the chain are derived and ranked as indicated in the table. The present pump model is not integrated into a planner (the ranked subgoals are printed out on the terminal); but the application to planning is evident, and we hope eventually to incorporate these techniques into the NOAH system (see Section III.B.2).

d. The RESTRICTIONS Function

The function (RESTRICTIONS part1 part2) computes all the FM restrictions imposed on part1 with respect to part2. (RESTRICTIONS part1 part2) is equal to (OPPOSITE (RESTRICTIONS part2 part1)). The answer is returned as a list of restricted directions, a subset of (+X -X +Y -Y +Z -Z). RESTRICTIONS could be

Table 5: Preconditions Required to Undo Restrictions

Probe Direction*	Type of CONNECTION	Precondition to Undo Restriction
Away	Restriction imposed by a fastening	1--Screw out fastener
Away	CONNECTION of type SCREWS-IN	2--Screw out the screwable part
Away	MESHES or SPINDLES	3--Move the parts in the direction permitted by MESH or SPINDLE
Toward	ABUTS	4--Move one part laterally with respect to the other. (The best direction is not determined.)
Toward	Fastened or MESHES or SPINDLES or SCREWED-IN	5--Action 1, 2, or 3 as appropriate, followed by action 4.

*Away means that the parts only need to be disengaged or unfastened to be free to move in the desired direction. Toward means that the parts would physically obstruct each other if moved in the desired direction.

computed by probing part1 in each of the six directions and seeing if it moves part2, as in MOVE above. However, this approach, aside from being inefficient, does not generalize to a representation in which arbitrary directions are allowed.

Another possible approach to computing RESTRICTIONS would be to find all paths of CONNECTIONS between part1 and part2, and to propagate all restrictions through each of these paths, finally taking the union of the restrictions imposed by all such paths to arrive at the answer. This method does generalize to the case of allowing arbitrary directions (assuming that there is a method of propagating restrictions through a particular CONNECTION); but it can run into a combinatorial explosion of the number of paths in a complicated device.

The algorithm actually used to compute RESTRICTIONS is a variant of the method of computing all paths, the second of the above methods. The algorithm takes the form of a search through a graph structure in which the nodes are parts and the edges CONNECTIONS. The search fans out from part1 and part2 simultaneously; FM restrictions on part1 and part2 are propagated to other nodes as the search proceeds. The node chosen for expansion at each step is the one that imposes the most restrictions on its "source" part (either part1 or part2). Whenever the search fanning out from part1 and part2 meets at some intermediate node, a path is completed from part1 to part2 and new restrictions on those parts may be imposed. The search is terminated immediately as soon as it is found that part1 and part2 are rigidly linked--i.e., have no freedom of movement with respect to each other. This search algorithm remembers the restrictions imposed by a particular connection (so they do not have to be computed more than once), allows those restrictions to be "used up" as the search progresses through a given CONNECTION, and deletes the connection from the search when its restrictions are completely "exhausted".

Still another method of computing FM restrictions that we have not implemented, but which we plan to investigate further, entails representing the FM between two parts as a bounded linear constraint on their coordinates. Then a standard linear programming algorithm might be used to find the constraints on the positions of a particular pair of parts.

e. An Example

We will now give a brief example of a dialog between the user and a system that models the air compressor pump. For the purposes of this example, the model's attention is confined to a subassembly of the pump consisting of the crankcase, head gasket, head, head cover, filter, filter-screen, long head bolts, short head bolts, and head-bolt spacers, as shown in Figure 43. In this example, minor parts such as the bolts, gaskets, and spacers are represented as individual parts in their own right, and enter into

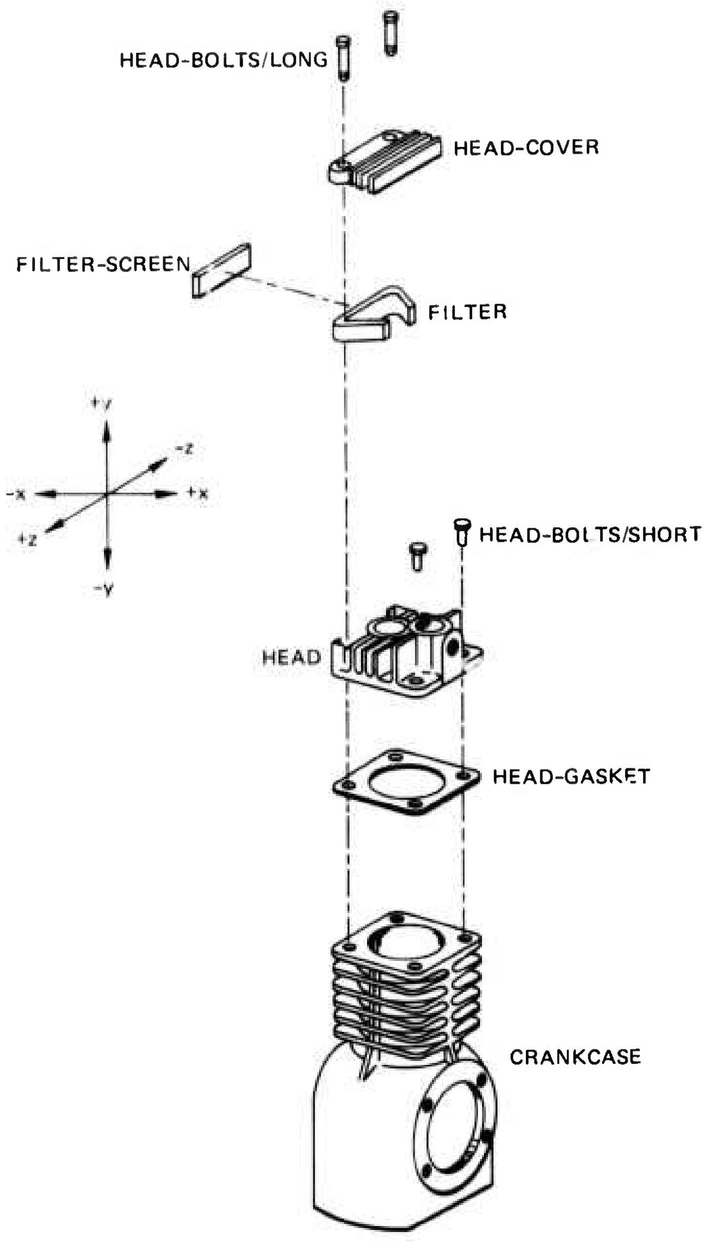


FIGURE 43 A SUBASSEMBLY OF THE PUMP UNIT

the FM computations as such. However, the model can also be made to ignore these parts and represent their effects abstractly in terms of fastenings.

The function BEGIN resets the model to the fully-assembled configuration and goes into an interactive mode in which it accepts model-changing commands and queries. The system uses the QLISP context mechanism, and so is generally capable of representing a tree of various hypothetical world states. For simplicity, BEGIN hides this feature by maintaining a "current" context, pushing this context whenever the state of the model is changed, and answering queries with respect to the current state. In the transcript that follows, the user's inputs are in upper case and always follow prompt characters (← or →); the system's replies are usually in lower case and at the left margin; explanatory comments that were added for the purposes of this discussion are indented and enclosed in brackets [ ].

```
←INITPUMP)
(model initialized)        [Initializes the components and CONNECTIONS of
                           the pump.]
←BEGIN)
(state of pump reset)

enter commands:
→ RESTRICTIONS HEAD-COVER CRANKCASE
(-X +X -Y +Y -Z +Z)        [They are rigidly attached]
```

```
→  SCREW-OUT HEAD-BOLTS/LONG

ok

→  RESTRICTIONS HEAD-COVER CRANKCASE

(+X -X -Y +Z -Z)

→  SCREW-OUT HEAD-BOLTS/SHORT

ok

→  RESTRICTIONS HEAD-COVER CRANKCASE

(-Y)

→  TIGHTEN HEAD-BOLTS/LONG

ok

→  TIGHTEN HEAD-BOLTS/SHORT

ok

→  MOVE FILTER +Y CRANKCASE

subgoals derived from path:

 ((2 screw-out HEAD-BOLTS/LONG) (5 move HEAD-COVER (-Y) HEAD-BOLTS/LONG ,
move HEAD-COVER (+X -X +Z -Z) HEAD-BOLTS/LONG) (4 move FILTER
(+X -X +Z -Z) HEAD-COVER))

Freedom of movement does not exist
```

[now the user plays the part of the planner and chooses an appropriate subgoal. The number at the beginning of each subgoal indicates its relative difficulty to achieve according to the table in section (c.) above. The first alternative, headed by "2", is derived from the CONNECTION between the HEAD-BOLTS/LONG and the CRANKCASE. The next, headed by "5", comes from (CONNECTION HEAD-BOLTS/LONG

```
                                        HEAD-COVER).  The last, headed by "4", comes from
                                        (CONNECTION FILTER HEAD-COVER).]
→ SCREW-OUT HEAD-BOLTS/LONG
                                        [The user chooses the first subgoal ...]
ok
→ MOVE FILTER +Y CRANKCASE  [... and tries again.]
forming new chunk of moved parts consisting of
 (HEAD-BOLTS/LONG HEAD-COVER FILTER FILTER-SCREEN) ...
                                        [The filter-screen gets carried along because it
                                        sticks to the filter.]
enter list of other moved parts that form new connections with TABLE:
→ (HEAD-COVER)                          [The model lacks the geometric information to
                                        figure this out for itself.]
forming new chunk of stationary parts consisting of
 (CRANKCASE HEAD HEAD-BOLTS/SHORT HEAD-GASKET HEAD-SPACERS TABLE) ...
new loose ends are:
 ((CONNECTION FILTER HEAD) (CONNECTION FILTER HEAD-SPACERS) (CONNECTION
HEAD HEAD-COVER) (CONNECTION HEAD-COVER HEAD-SPACERS) (CONNECTION
CRANKCASE HEAD-BOLTS/LONG) (CONNECTION HEAD HEAD-BOLTS/LONG) (CONNECTION
HEAD-BOLTS/LONG HEAD-SPACERS) (CONNECTION HEAD-BOLTS/LONG HEAD-GASKET)
(CONNECTION FILTER-SCREEN HEAD)) ...
warning -- the following parts may become unsupported:
 (HEAD-BOLTS/LONG FILTER-SCREEN HEAD-COVER) ...
ok
→ STOP
done.
```

f. Continuing Work

The simple pump model described above has served as a test of freedom-of-movement as a modeling tool. We are working to develop a better understanding of the semantics of freedom of movement and more general representations of FM restrictions. These concepts will be incorporated in the model of the compressor and in models of other devices that may be dealt with in the CBC project.

We are beginning to examine the relationships between the geometric device models being used for vision and display (see Section II.C) and the more symbolic models used for assembly-disassembly planning, with an eye toward automating, as much as possible, the creation of similar reresentations for new devices. Perhaps the user can characterize a new device by the nature of the CONNECTIONs between its parts (as in the pump model described above) and the general shapes of the parts, using some visual input where necessary. If the system had a three-dimensional shape model (even a crude one) and a classification of inter-part CONNECTIONs (ABUTS, SPINDLES, and the like), it could derive freedom of movement and access information. This in turn could yield planner preconditions for assembly and disassembly actions.

These investigations raise many of the issues prevalent in current A.I. research relating to "world knowledge" --

how it is acquired, how to represent it internally, how to retrieve, organize and update it. In the CBC domain, we will especially confront the need to represent the same device from different points of view for different purposes such as display, vision, assembly/disassembly planning, and natural language interaction. Particular solutions to these problems in the CBC domain may shed some light on the larger issues involved.

2. Diagnosis of Equipment

a. Background

The use of computers to help diagnose malfunctions in equipment and to help diagnose diseases in humans already has an extensive history. Notable examples are systems that diagnose blood infections [14], systems that locate faults in digital circuits [45], and systems that locate faults in internal combustion engines [46].

Several approaches to computer diagnosis have been studied, and we shall briefly mention three of them here. Two approaches, using "truth tables" and using "decision trees," have parallels in pattern recognition technology. A third, the "rule-based" approach, stems from heuristic programming techniques in Artificial Intelligence.

The so-called truth table method entails collecting a number of simultaneous measurements from the system being diagnosed. We might represent these measurements by an n-dimensional vector X, where n is the number of measurements made. Then an analysis is made of the effects on the vector X of various faults that might occur in the system. In the simplest case each possible fault has a unique effect on X. For example, a perfectly functioning system might give rise to the vector Xo. Each different fault gives rise to a distinct vector X; for example, the i-th fault gives rise to vector Xi. This association between faults and vectors can be stored in a table. In diagnosis, one enters the table using the measurements X to find the associated fault.

In more complex cases the mapping between faults and measurements is not one-to-one. This presents a difficulty when the same measurement vector X could have been caused by a number of different faults. In the latter case, common in medicine, it is customary to collect a large amount of data in order to estimate the probabilities p(X/fault) and then to use Bayes' rule to calculate the a posteriori probabilities p(fault/X). Standard decision theory techniques can then be used to decide on the fault given any measurement vector X.

The decision tree approach locates the fault by using measurements in sequence instead of simultaneously. It is like the game of "twenty questions." Depending on the result of the first measurement, another measurement is selected. Its result in turn determines another measurement, and so on. The alternative measurements can be pictured as a tree. The identities of the faults are located at the tips of the tree, i.e., after the final measurements in each sequence. On the average, this approach requires fewer measurements than the truth table approach, but like all sequential procedures has little tolerance for errors in measurement.

Several problems with these "classical" methods have motivated a search for some alternative approach to diagnosis. In particular, we might mention the following:

(1) Data gathering. The data needed for the truth table method are very often statistical. Large amounts must be gathered under carefully controlled conditions. Much more easy to gather are the "distilled", "judgemental" data used by skilled diagnosticians. Often this information is in the form of simple rules like: "If symptom X is present, then the trouble is likely to be Y." An argument can be made that the diagnostician himself uses rules of this sort in his own information processing procedures leading to diagnosis, and,

in particular, does not explicitly use tables or Bayes' rule. Presumably it would be more effective and efficient to extract these useful rules by interviewing skilled practitioners rather than to gather raw statistical data.

(2) Extendibility. Statistical data are not easily extendible to slightly different problem areas or to situations in which new measurements are discovered to be useful. Each new problem area or advance in technique would have to be accompanied by another extensive statistical analysis. Decision trees are similarly hard to modify to include a new measurement. Generally, the whole tree must be redesigned. We would like a technique that can easily grow to include new information.

(3) Interaction. The classical methods do not allow a human expert to interact with the diagnostic process in any flexible way. In the truth table method, once the measurements are input, the process takes complete control

until a decision is announced. A human user cannot interrupt it to contribute any additional information that might be relevant. With decision trees, the human user's role, if any, is limited to answering questions about the results of any of the measurements that are not made automatically. The human cannot volunteer any new information, but must patiently wait until and if it is requested. We desire a system that can be interrupted when the user thinks he has important additional information.

(4) Transparency. The classical methods arrive at decisions that cannot always be explained to the satisfaction of a skilled user. This is because the user is likely to think in terms of judgemental concepts or rules as described earlier. When queried about a decision, all the classical methods can do is state obscure statistical facts or present details about the structure of underlying tables or trees. We want a system that "thinks" like a practitioner so that it

can explain its chain of deductions in terms understandable to a practitioner.

In order to meet these criteria, we have been thinking about designs stemming from AI research on rule-based systems. A good example of such a system (even though it does not meet criterion 3 above) is the MYCIN medical diagnosis system of Shortliffe [14]. Our tentative design outline, to be described in the next section, is similar in spirit to MYCIN, although the mechanisms of operation are different and it will be an interactive system.

b. System Design

Although our work on diagnosis is still in an early stage, we have developed a tentative design and have done some initial programming to test it.

The diagnosis process begins with a statement of a "presenting symptom" from the apprentice. Based on this information a list of hypotheses about probable faults that might be causing the symptom is generated. A probability is associated with each of these faults.

The fault hypotheses are generated by using rules of the form: "If the symptom is X, the fault is likely (probability = p) to be Y." Actually each rule is a small program that can perform any necessary computations required to set up the hypotheses. Our idea is that the rules will be written in close collaboration with skilled diagnosticians.

In diagnosing a large system there are many levels of symptoms and faults. Suppose, for example, that we are diagnosing a malfunctioning automobile. A symptom might be that the engine does not start. A "high-level" explanation might be that the ignition system is faulty. At a somewhat lower level of the hierarchy, the symptom might be that the ignition system does not work, and the fault might be that the spark is too weak. Continuing down the hierarchy, we might finally learn that the spark plugs are dirty.

Our diagnosis system is being designed at the start to work on a hierarchical basis. Each symptom is presumed to be caused by faults of an appropriate level, which in turn describe symptoms caused by lower level faults.

After a list of fault hypotheses is generated, one of them is chosen as worthy of testing. Currently, we are selecting the most probable hypothesis, but we are open to using a more sophisticated criterion. After a hypothesis is selected, we refer to another set of rules telling us how to test for it. These rules are called "handlers." They are of the form: "If Y is a suspected fault, use test T, and, depending on the outcome, modify the probability of Y." Again, the rules are small programs that can perform any computations needed to help specify the test and interpret its results. A choice between competing handlers for the same hypothesis can be made on the basis of benefit/cost ratios of the associated tests. The result of applying a handler is to modify the probabilities of the hypotheses. (A test may give information about hypotheses other than the one for which it was chosen; we will use "demons" to catch such additional information.) We now loop back through the main program, select the now most-probable hypothesis, select a handler for it, and so on. A trap condition allows us to exit whenever the probability of one of the hypotheses exceeds some threshold. This hypothesis is then decided upon as the fault.

Depending on the apprentice's skills and desires, this fault decision may or may not be the final answer to his diagnostic problem. If he needs more detailed information, he may re-enter the fault just computed by the program into the system as a symptom. The process then uses a set of finer-grained rules to give a more detailed explanation of the fault.

We envision the system working in the hierarchical fashion just described, punctuated by outputs to the apprentice and restarting. Even so, there may be reasons why the system might want to descend a few levels on its own to help make more definite conclusions about higher level faults. That is, before deciding definitely that a higher level hypothesis is correct, the system may on its own test some of the subhypotheses explaining the higher level one as a way of confirming or refuting the higher level one.

We have written an initial QLISP program to test some of these ideas and have applied it to problems of diagnosing some simple electric circuits. After we finally decide on a more complex piece of equipment to replace the air compressor, we intend to implement a much larger diagnosis system for that equipment. We also still must face several important questions involving multiple faults and intermittent faults. We will be postponing the matter of how to repair faults until we make substantially more progress on the diagnosis problem.

Our description of the operation of the hypothesis handlers glossed over the question of how probabilities are to be modified as a result of the tests. This is an important and nontrivial problem. We have adopted a Bayesian approach to this matter that is discussed in some detail in the next subsection.

C. Alternative Forms of Bayes' Rule for Diagnosis

1) Introductory Remarks

The use of Bayesian decision theory is popular in medical diagnosis [47,48,49], and has a natural appeal in any diagnosis problem in which one has suggestive but not compelling evidence to support various hypotheses. As Shortliffe has pointed out [50], direct use of Bayes' theorem is not without serious drawbacks. These problems are particularly severe when there is no cause-and-effect model, and when correlations between the joint occurrence of a set of evidence and the hypotheses must be estimated from data. Shortliffe and Buchanan [51] give additional arguments for the inappropriateness of Bayes' theorem in rule-based diagnosis systems, and advocate their version of confirmation theory that substitutes certainty factors for conditional probabilities.

This subsection presents several variants of Bayes' theorem, some of which are very similar to Shortliffe's certainty factor formulas. The variants are appropriate under different circumstances. We begin with the simplest case of a single hypothesis and binary-valued evidence.

ii) Bayes' Rule for Events

Let H be a hypothesis that is either true or false, and let E be a piece of evidence that either is or is not present. We view E and H as random events, writing P(E) as the a priori probability of the occurrence of event E, P(H) as the a priori probability of event H, and P(E,H) as the joint probability of E and H, i.e., the probability of the event $E \wedge H$. For the nonoccurrence of E or H, we write

$$Q(E) = P(\overline{E}) = 1-P(E) \tag{C1}$$

and

$$Q(H) = P(\overline{H}) = 1-P(H) \quad . \tag{C2}$$

For the joint probability function, we must know three of the four probabilities $P(E,H)$, $P(E,\overline{H})$, $P(\overline{E},H)$ and $P(\overline{E},\overline{H})$. By the law of total probability, $P(E)$ and $P(H)$ can be derived from the complete joint probability function by

$$P(E) = P(E,H) + P(E,\overline{E}) \tag{C3}$$

$$P(H) = P(E,H) + P(\overline{E},H) \quad . \tag{C4}$$

The conditional probability $P(E\backslash H)$ is defined as

$$P(E|H) = \frac{P(E,H)}{P(H)} \quad . \tag{C5}$$

Similarly,

$$P(H|E) = \frac{P(E,H)}{P(E)} \tag{C6}$$

[illegible]

with analogous formulas for $P(E\backslash\bar{H})$ and $P(H\backslash\bar{E})$. Now, typically, we either know or can estimate both $P(E\backslash H)$ and $P(E\backslash\bar{H})$, we obtain the evidence E, and we wish to update the probability of H from the a priori value P(H) to the a posteriori value $P(H\backslash E)$. Bayes' rule, which follows immediately from the above formulas, provides the desired answer:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E|H)P(H) + P(E|\bar{H})P(\bar{H})} \quad . \tag{C7}$$

iii) Some Useful Variants

Define the likelihood ratio by

$$\Lambda(E|H) = \frac{P(E|H)}{P(E|\bar{H})} \tag{C8}$$

and let the ratio O(H) be the odds favoring hypothesis H,

$$O(H) = \frac{P(H)}{P(\bar{H})} = \frac{P(H)}{Q(H)} \quad . \tag{C9}$$

Then we can rewrite Eq. (C7) as

$$P(H|E) = \frac{\Lambda O}{\Lambda O + 1} \quad . \tag{C10}$$

Thus, the a posteriori probability can be determined from the likelihood ratio and the a priori odds. Alternatively, if we use the fact that

$$P(\bar{H}|E) = 1 - P(H|E) = \frac{1}{\Lambda O + 1}$$

and define the a posteriori odds by

$$O(H|E) = \frac{P(H|E)}{P(\bar{H}|E)} \tag{C11}$$

then we obtain the simple formula

$$O(H|E) = \Lambda(E|H)\ O(H) \tag{C12}$$

This formula is sometimes called the odds-likelihood formulation of Bayes' rule [52]. It shows how the odds favoring a hypothesis change when new evidence is obtained. For our purposes, it is perhaps the most useful form of Bayes' rule. It suggests a simple recursive updating procedure that will be elaborated in Subsection V below. To recover the probabilities from the odds, we merely use the formulas

$$P(H) = \frac{O(H)}{O(H) + 1} \tag{C13}$$

and

$$P(H|E) = \frac{O(H|E)}{O(H|E) + 1} \quad . \qquad \text{(C14)}$$

iv) Objections and Problems

There are two basic problems in using these results. The traditional objection concerns the a priori probabilities. If H is the hypothesis of an event that has never been known to occur naturally, then even though we may be able to force H to occur and thus determine P(E\H), there may not be any objective way to determine P(H). Although this can be viewed as a philosophical objection to the whole approach, the degree of seriousness of the problem has often been overstated. Even if H is a rare event, so that P(H) is small (but not zero), P(H\E) can be significantly large if the evidence E is sufficiently strongly linked to H. In particular, we see from Eq. (10) that P(H\E) will be near unity if ΛO is much greater than unity, i.e., if $\Lambda >> 1/P(H)$ when P(H) is small.

This leads is to a second and more serious problem. It is obvious that the evidence is effective in confirming H if and only if it is much more likely to be obtained when H is true than when H is not true. Unfortunately, while it may be possible to get a very reasonable estimate for P(E\H), it is often very hard to estimate $P(E\backslash\overline{H})$. Consider, for example, the case in which H is the hypothesis that a particular car is out of gasoline, and the evidence E is that the car will not start. Clearly, $P(E\backslash H)$ is very near unity, but what is $P(E\backslash\overline{H})$? The problem is that $\overline{H}$ is not a simple hypothesis but a compound hypothesis. There are many possible reasons for the car not starting. While we may be able to estimate the probability of E given any one of them, we may find it hard to account for their interactions, and we may not even be able to think of all of the possible explanations.

The problem we face here is not unlike the problem that Shortliffe faces in assigning confidence values to his diagnosis rules. Suppose that we have a rule such as "If the car will not start, then there is suggestive evidence that the car is out of gas." Should we assign this rule a confidence of 0.1 (weakly suggestive)? 0.5 (suggestive)? 0.8 (strongly suggestive)? Shortliffe relies on the expert to make this decision. Presumably, the expert has encountered many cars that will not start, and has a rather good idea of what can be concluded.

When necessary, we adopt a similar attitude toward the likelihood ratio. Even though the expert may be hard pressed to give good numerical estimates for either $P(E\backslash H)$ or $P(E\backslash\overline{H})$, we boldly assume that the expert can quantify subjective feelings about the strength of the relation between a hypothesis and some evidence by giving reasonable numerical estimates for the log-likelihood ratio

$$L(E|H) = \log_b \Lambda(E|H) \quad . \tag{C15}$$

The log-likelihood ratio is of interest for several reasons. It has been found to be advantageous in at least some experiments in which people are asked to estimate likelihood ratios [53]. Mathematically, the use of logarithms turns the odds-likelihood formulation of Bayes' rule into an additive form

$$\log O(H|E) = L(E|H) + \log O(H) \tag{C16}$$

Thus, L(E\H) is an additive measure of the "information" that E gives about H. If L is positive, H is more likely; if L is negative, H is less likely; if the evidence is indifferent, L = 0. The actual numerical value of L depends on the choice of the base for the logarithm. In most of our work we have used a base, b, of 1.5. With this arbitrary choice, the range from -10 to +10 for L corresponds to a range of about 1/60 to 60 for Λ.

v) Sequentially Acquired Evidence

Suppose that we have previously obtained n - 1 pieces of evidence E1, ..., En-1 and that we know P(H\E1, ..., En-1), or, equivalently, the odds O(H\E1, ..., En-1). We can treat these as prior probabilities (or odds) when a new piece of evidence En is obtained. Thus, we can rewrite Eq. (C12) as

$$O(H|E_1, \ldots, E_n) = \Lambda(E_n|H,E_1, \ldots, E_{n-1})O(H|E_1, \ldots, E_{n-1}) \tag{C17}$$

where

$$\Lambda(E_n|H,E_1, \ldots, E_{n-1}) = \frac{P(E_n|H,E_1, \ldots, E_{n-1})}{P(E_n|\bar{H},E_1, \ldots, E_{n-1})} \tag{C18}$$

This shows how O(H\E1, ..., En-1) can be updated recursively to yield O(H\E1, ..., En). Alternatively, we can keep all n pieces of evidence grouped together and rewrite Eq. (C12) as

$$O(H|E_1, \ldots, E_n) = \Lambda(E_1, \ldots, E_n|H)O(H) \tag{C19}$$

where

$$\Lambda(E_1, \ldots, E_n|H) = \frac{P(E_1, \ldots, E_n|H)}{P(E_1, \ldots, E_n|\bar{H})} \quad . \tag{C20}$$

However we write these expressions, one fact always remains--the likelihood ratio depends on all n pieces of evidence. In that idyllic world in which the pieces of evidence are conditionally independent, so that P(En\H, E1, ..., En-1) = P(En\H) and P(En\$\bar{H}$, E1, ..., En-1) = P(En\$\bar{H}$), we obtain

$$O(H|E_1, \ldots, E_n) = \Lambda(E_n|H)O(H|E_1, \ldots, E_{n-1}) \tag{C21}$$

with

$$\Lambda(E_n|H) = \frac{P(E_n|H)}{P(E_n|\bar{H})} \quad . \tag{C22}$$

Alternatively, we have

$$O(H|E_1, \ldots, E_n) = \Lambda(E_1, \ldots, E_n|H)O(H) \tag{C19}$$

with

$$\Lambda(E_1, \ldots, E_n|H) = \prod_{i=1}^{n} \Lambda(E_i|H) \tag{C23}$$

$$= \prod_{i=1}^{n} \frac{P(E_i|H)}{P(E_i|\bar{H})} \quad . \tag{C24}$$

vi) Combining Independent A Posteriori Probabilities

Suppose that we had some good method for estimating the probability of hypothesis H given just the single piece of evidence E. Let

$$p_i = P(H|E_i) \tag{C25}$$

$$q_i = P(\bar{H}|E_i) = 1 - p_i \tag{C26}$$

and

$$o_i = \frac{p_i}{q_i} \quad . \tag{C27}$$

Then the likelihood ratio can be written as

$$\begin{aligned} \Lambda(E_i|H) &= \frac{P(E_i|H)}{P(E_i|\bar{H})} \\ &= \frac{P(H|E_i)P(E_i)/P(H)}{P(H|E_i)P(E_i)/P(\bar{H})} \\ &= \frac{o_i}{O(H)} \end{aligned} \tag{C28}$$

Substituting this in Eqs. (C24) and (C19), we obtain

$$O(H|E_1, \ldots, E_n) = \left[\prod_{i=1}^{n} \frac{o_i}{O(H)}\right] O(H) \tag{C29}$$

$$= \frac{o_n}{O(H)} O(H|E_1, \ldots, E_{n-1}) \quad . \tag{C30}$$

Note that the a posteriori odds increase whenever the odds based on En alone exceed the a priori odds. It follows that the a posteriori probability increases whenever P(H\En) > P(H). If we express the odds in terms of probabilities, we obtain

$$P(H|E_1, \ldots, E_n) = \frac{1}{1 + \left[\prod_{i=1}^{n} \frac{Q(H|E_i)P(H)}{P(H|E_i)Q(H)} \right] \frac{Q(H)}{P(H)}}$$

$$= \frac{1}{1 + \frac{Q(H|E_n)Q(H|E_1, \ldots, E_{n-1})P(H)}{P(H|E_n)P(H|E_1, \ldots, E_{n-1})Q(H)}} \qquad \text{(C31)}$$

Note that for the special case n = 1 we merely obtain the identity P(H\E1) = P(H\E1). The case n = 2 gives

$$P(H|E_1,E_2) = \frac{1}{1 + \frac{Q(H|E_2)Q(H|E_1)P(H)}{P(H|E_2)P(H|E_1)Q(H)}} \qquad \text{(C32)}$$

This can actually be interpreted as the general case if we think of E1 as the collection of all of the old evidence and E2 as the new piece of evidence. The graph in Figure 44 shows how P(H\E1,E2) changes with P(H\E2) for the special case P(H) = 1/2.

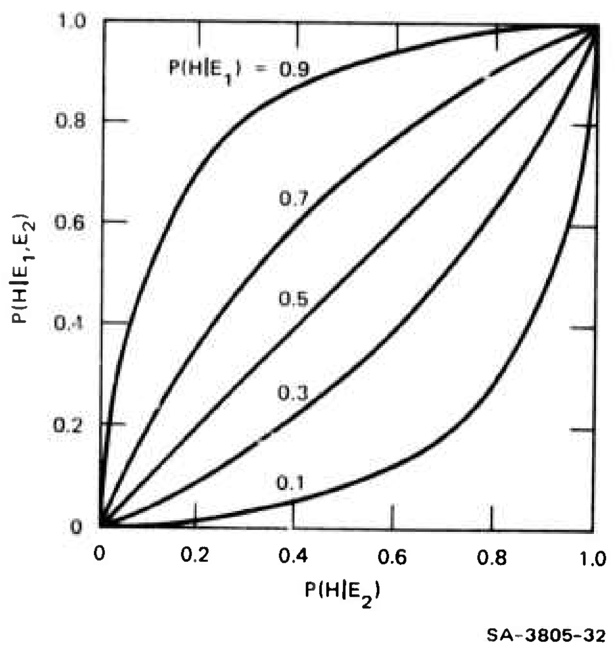


FIGURE 44 THE EFFECT OF NEW INDEPENDENT EVIDENCE ON THE A POSTERIORI PROBABILITY OF H

vii) Relation to Shortliffe's Combining Formula

Additional variants can be obtained by introducing a quantity C that is analogous to--though not exactly the same as--Shortliffe's certainty factor CF. In general, for any probability P we define C as

$$C = 2P - 1 \quad . \tag{C33}$$

Thus, as P varies from 0 to 1, C varies from -1 to 1. After a little algebraic manipulation, we can express Eq. (C32) as

$$C(H|E_1, E_2) = \frac{C_1 + C_2 - C_0(1 + C_1C_2)}{1 + C_1C_2 - C_0(C_1 + C_2)} \tag{C34}$$

where

$$C_1 = 2P(H|E_1) - 1 \tag{C35}$$

$$C_2 = 2P(H|E_2) - 1 \tag{C36}$$

and

$$C_0 = 2P(H) - 1 \quad . \tag{C37}$$

It is instructive to consider some special cases. If all of the C's are strictly between -1 and 1, then it is easy to show that

$$C(H|E_1,E_2) \to 1 \quad \text{as} \quad C_1 \to 1$$

$$\text{or as} \quad C_2 \to 1$$

$$\text{or as} \quad C_0 \to -1$$

and that

$$C(H|E_1, E_2) \to -1 \text{ as } C_1 \to -1$$

$$\text{or as } C_2 \to -1$$

$$\text{or as } C_0 \to 1 \quad .$$

The limiting behavior for C1 and C2 seems quite proper, but the behavior for C0 at first seems paradoxical. If we are virtually certain a priori that H is false (C0 = -1), why should we be convinced that H is true a posteriori? The explanation is that if P(H) is approaching zero but P(H\E1) or P(H\E2) stay bounded away from zero, then the evidence in favor of H is extremely convincing.

For the special case P(H) = 1/2 we obtain

$$C(H|E_1, E_2) = \frac{C_1 + C_2}{1 + C_1C_2} \qquad \text{(C38)}$$

or, alternatively,

$$C(H|E_1,E_2) = C_1 + C_2(1 - C_1) + \left[1 - C(H|E_1,E_2)\right] C_1C_2 \approx C_1 + C_2(1 - C_1)$$

if $C(H\backslash E1,E2)$ is near unity. This latter expression is reminiscent of Shortliffe's formulas for updating measures of belief and disbelief for sequentially acquired evidence. Thus, to first order, we can think of Shortliffe's formulas as an approximation to Eq. (C38), which in turn is for the special case of independent evidence with $P(H) = 1/2$.

However, the approximation is not good if C1 and C2 have opposite signs, corresponding to conflicting evidence. For example, suppose that $C1 < 0$ and $C2 > 0$. In this circumstance, Shortliffe would say that E1 favors disbelief in H and E2 favors belief in H, and would write

$$MB(H, E_1) = 0$$

$$MD(H, E_1) = -C_1$$

$$MB(H, E_2) = C_2$$

$$MB(H, E_2) = 0$$

$$MB(H, E_1 \& E_2) = MB(H, E_2) = C_2$$

$$MD(H, E_1 \& E_2) = MD\ (H, E_1) = -C_1 \quad ,$$

provided that neither C2 nor -C1 is unity. The certainty factor is obtained as the difference of these measures of belief and disbelief:

$$CF(H, E_1 \& E_2) = C_1 + C_2 \quad .$$

Note that this leads to a curious discontinuity in the behavior of the certainty factor. If C1 = -1/2, then the largest CF can become as C2 approaches unity is 1/2; however, in the special case C2 = 1, Shortliffe sets MD(H, E1 & E2) to zero and obtains CF = 1. This awkward limiting behavior is avoided by Eq. (C38) (and by the other, more general formulas we have derived), in which C(H\E1,E2) approaches unity continuously as C2 approaches unity. This will be true for any value of C1 except C1 = -1. In general, one must expect anomalous behavior when both conflicting conclusions are certain, but there seems to be no reason to accept discontinuities in noncontradictory situations.

D. Vision

1. Introduction

The vision modules associated with the pointing system described in Section II.C were designed for limited and well defined purposes. Our long-term objective is considerably more ambitious: the construction of a visual information gathering facility that may be called on by other subsystems to answer questions from the apprentice, monitor his performance, and verify and update the knowledge base. This facility will consist of a large number of general and special purpose vision modules coordinated by a problem solving executive. This executive will use knowledge of the task context, of the perceptual domain, and of the available perceptual capabilities, to plan effective strategies for fulfilling information requests. This section describes two major research efforts, one on perceptual strategies, the other on scene understanding, directed towards this long range objective.

The techniques required for fulfilling information requests may range from a simple table lookup to a major analysis of the entire scene. Certain common requests may be handled directly by passing them to a packaged module such as a tool recognizer. However, many tasks will undoubtedly require planning at a much lower level. Consider a question such as "is the flywheel on backwards." It so happens that the flywheel on our compressor is painted only on its front surface. Knowing this fact, the particular question posed above can be answered relatively easily by testing color at the predicted image location of the flywheel. (The prediction could be based on the same compressor model used for pointing.) Obviously it is impossible to provide canned strategies of this type for all questions the apprentice might reasonably ask.

The issues entailed in planning perceptual strategies are fundamental ones that arise in every complex perceptual domain regardless of the particular modules involved. First the available options for accomplishing the task must be assembled in a planning graph. Second, the options must be organized into a plan of action based on relative cost and likelihood of achieving desired results. Third, the plan must be monitored during execution to detect departures from initial planning assumptions and, if necessary, be revised. For several years, we have been investigating these issues in the context of strategies for finding objects in room scenes [37, 54].

There are similarities between strategies for object finding and those for question answering; the same techniques that select features for distinguishing a chair from other room objects can be used for distinguishing the front and back of a pulley. In the next subsection we describe an implemented system that can plan cost-effective information gathering strategies in a variety of domains.

The second area of research concerns scene understanding. Machinery is one of the most difficult scene analysis domains. Components on a given mechanical assembly often cannot be distinguished by any local characteristics except their detailed shape. Moreover, visual characteristics may vary widely over a generic class of components performing the same function (e.g., carburetors). Part identification is thus likely to result in ambiguities when the parts are viewed in isolation. Fortunately, a rich set of constraints governs relationship of parts to each other in a complete assembly. These constraints may arise from consideration of the function or operation of the parts, or from methods of fabrication or assembly. For example, carburetors are connected to the intake manifold, a fuel line, and a throttle linkage. The absence of neighboring parts that could reasonably assume these interpretations makes it unlikely that a given part is a carburetor. Our goal is to use such general knowledge about a class of equipment, (e.g., automobile engines) to resolve interpretation ambiguities on individual parts, eliminating the need for detailed structural models of specific units.

A framework for performing this type of reasoning about scenes was described in last year's final report. Since then, the design has been refined, implemented, and tested in the room scene domain. Room scenes were used for experimental convenience in testing the basic reasoning system because representations for both local features and global relations were already available. Comparable representations are currently being developed for the mechanical equipment domain with local interpretations based primarily on 3-D shape inferred from range data. This work is reviewed in a subsequent subsection.

2. Scene Analysis Plan Generation and Execution

We have designed and implemented a planning system that makes use of information about anticipated costs and reliabilities to choose the best course of action. Although the planner was designed and implemented as a core subsystem of a vision system that locates objects in office scenes [37, 55-57], we feel that the applicability of the process reaches further than the boundaries of scene analysis. Therefore, while the examples will mostly be chosen from problems in that domain, the presentation will be of a more general nature.

For artificial intelligence problem solvers, the real problem has traditionally been to generate any plan for performing the desired task; since the choices for solving the problem were generally limited, the choice of which subgoal to work on and subsequent execution has been the easy part. The planner to be described here generates a (relatively) complete plan for the analysis of a scene, but then, more importantly, organizes the plan so as to know what to work on and in what order. The problem at any point in execution is to know what to do next. An example of a simple plan for finding a chair is shown in Figure 45. We will go into detail about the generation of this plan below.

The planner first elaborates the goal into the subgoals required for its solution. These subgoals are elaborated in turn until no more remain. The result is an AND-OR planning graph with tip nodes that are executable subgoals. Next, the scorer organizes the graph so that the executor can find the best terminal node to begin evaluating. The plan executor executes the best

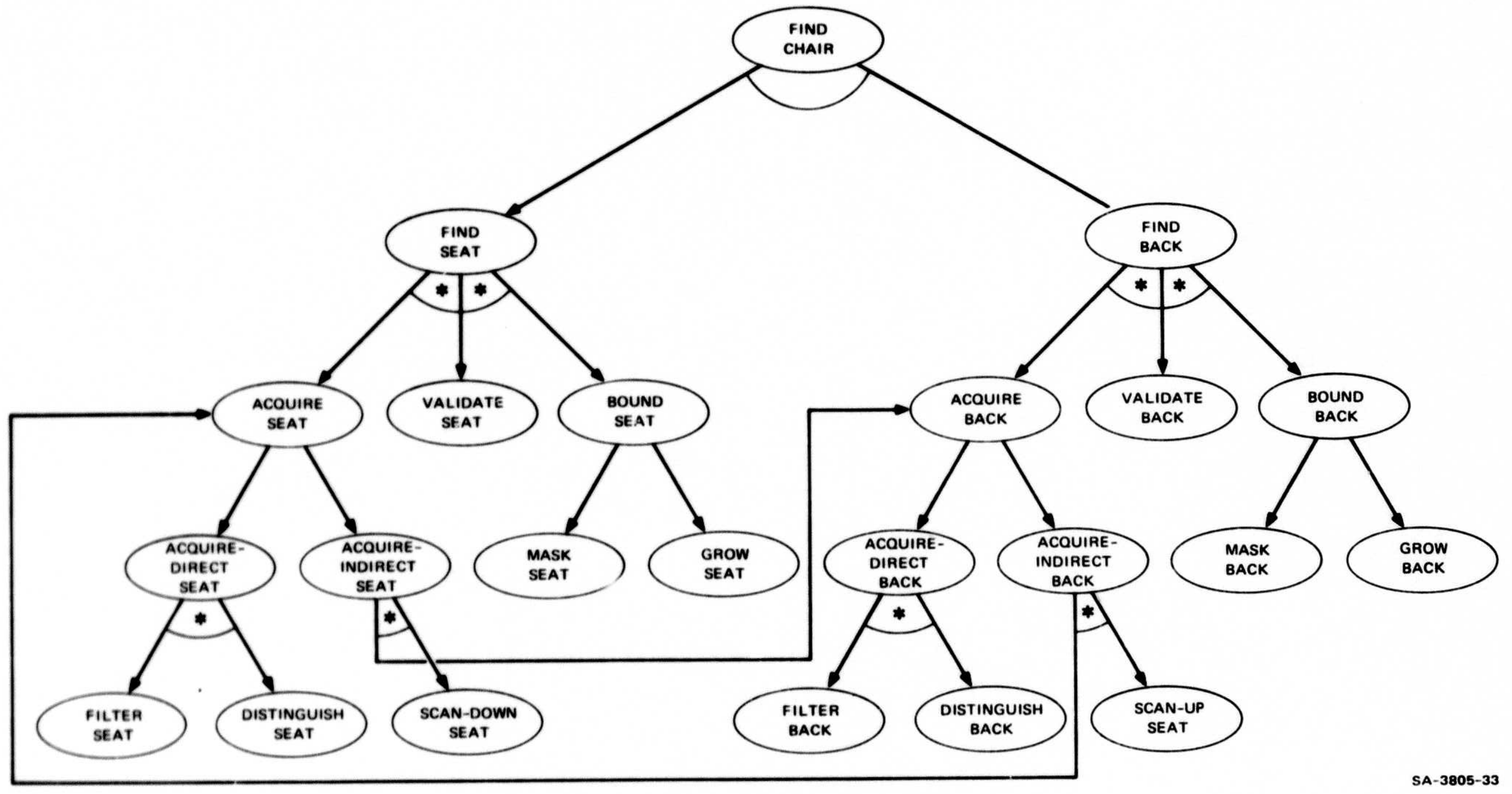


FIGURE 45 DIAGRAM OF PLAN FOR FINDING A CHAIR

subgoal and then propagates the success or failure of the goal through the original plan, in order to select subsequent subgoals. Execution continues until the top goal either succeeds or fails, at which time the process of planning and execution is complete.

a. Definitions

Before describing how the planner works, we will present some definitions and descriptions of strategies, plans, and elements of plans, the most important of which is the goal node. This node contains information about the satisfaction of the goal, and also contains (implicitly) the complete structure of the plan.

Goal - a representation of some state to be achieved. A goal is either directly realizable (i.e., executable), or is an intermediate goal, which is satisfied by satisfying its subgoals. An executable goal is a terminal goal.

The format of a goal is a list, the first element of which is usually the activity represented by the goal (e.g., FIND, VALIDATE). The second element is usually the object for which the goal is being executed. Succeeding elements (if any) usually contain information pertinent to the goal. For example,

```
(FILTER-WINDOW  DOOR
                (LAMBDA (X) (LIMITP X (FUNCTION HEIGHT)
                                       2.5 5.0))
                            (LIMITP X (FUNCTION SAT)
                                       .6 .75)))
                ((HEIGHT (6 . 8))
                 (SAT (7 . 12))))
```

is a typical goal. The activity is FILTER-WINDOW, a filtering program that passes a LISP predicate over a sampled window of the image; the object is DOOR; the next element is the LISP predicate to be used in filtering, and the last element is the internal representation of that predicate. The last two elements are particular to FILTER-WINDOW.

Subgoal List - a list of subgoals preceded by an operator from the set: AND, *AND, OR, or ↑. The operator indicates how solution of the subgoals relates to solution of the goal. AND and *AND imply that all the subgoals must be achieved to satisfy the goal; OR implies that only one of the subgoals must be satisfied. A *AND requires that the subgoals be achieved in sequence. ↑ is a dummy operator that is used when there is only one subgoal; it means that satisfying the subgoal is equivalent to satisfying the goal. This operator is used primarily to minimize the complexity of the subgoal lists. Some example subgoal lists are:

```
(AND (FIND SEAT)(FIND BACK)),
(*AND (ACQUIRE DOOR)(GROW DOOR)), and
(OR (ACQUIRE-DIRECT PICTURE) (ACQUIRE-INDIRECT PICTURE)).
```

The first example is the subgoal list for (FIND CHAIR); the second for (BOUND DOOR); the third for (ACQUIRE PICTURE).

Node - (or goal node) is a list consisting of the goal, the subgoal list for the goal, a set of parameter lists for scoring purposes, and the parents of the goal. For example,

```
((ACQUIRE DOOR) (OR (ACQUIRE-DIRECT DOOR)
                    (ACQUIRE-INDIRECT DOOR))
                ((.9) (.95) (.855) (50000.) (58500.))
                ((BOUND DOOR)))
```

could be the goal node for the goal (ACQUIRE DOOR).

Plan - the set of goal nodes generated for satisfying the initial goal. In its most general form, the plan is an implicit AND-OR graph, since the subgoals point to one another through their own subgoal lists. The graph structure appears when a goal has some other for a subgoal, which in turn has the first as one of its subgoals (although not necessarily as a direct descendant). For example, as shown in Figure 45, to find the seat of a chair it might be possible to first find the back and use that to localize the seat. However, to find the back, it might also be possible to first find the seat and then localize the back. This sort of reasoning lads to plans with loops.

Module - a module is a program or a statement describing how to satisfy a goal. In the case of executable goals, the module is an execution module and is the actual program to be executed. For nonterminal goals, the module is a planning module that describes how the goal is to be expanded into subgoals. A module may have various functions associated with it. Some common ones are the COSTFN (which computes the cost of execution of the module), and the PTFN (which computes the probability of success). In addition, the module may also have associated functions for deciding if a previously generated goal is equivalent, or related.

Cost - in the context of planning, the cost is the anticipated time that would be spent trying to satisfy a goal. In the case of bounded processes, such as filtering, the cost can be estimated rather closely. For less bounded processes, such as region growth, the cost is approximated from past experience and whatever relevant information may be available (for example, the expected size of the region to be grown).

Confidence - in the context of planning is the probability that the execution of the goal will have a good outcome. This is composed of two parts, the probability that the outcome will be good, given that the node succeeds, and the probability of success of the goal. Again, in cases where the process is well modeled, the estimates are relatively good, and where there is no good model, the estimates are based largely on past experience. Confidence is always between 1.0 and 0.0.

Score - the score of a node is the value of a function that computes a cost figure normalized by confidence. For example, cost divided by confidence is the measure that we use here; however, other functions are being experimented with.

b. Plan Elaborator

To create a plan, the plan elaborator (PE) expands the initial goal (the "top goal", which is typically to find some object, e.g., FIND(TABLE)) into the subgoals required for its solution. The subgoals are recursively elaborated in exactly the same way. If the PE cannot generate subgoals, it checks to see whether the goal is directly executable, and if so computes parameters (cost and confidence) for later use in scoring. If it is not directly executable and has no subgoals, the elaborator marks it for later deletion.

As new objects are noticed by the PE, their names are added to an "instance list". This list is used as a global "note-pad" to keep track of what is learned about the objects during execution of the plan. The initial entries into the instance list are taken from a globally maintained list of prototypes; for example, when the chair seat is noticed for the first time in the course of planning, the name is looked up on the list of prototypes. This entry is then added to the instance list. The prototype contains general information such as relations between the object and other objects, or previous plans that were created. This information is available to the planner during elaboration, and is also used by modules during execution as a storage medium containing up-to-date information about previously executed modules. For example, the chair seat prototype could contain the information that the chair seat is below the chair back. During execution of a plan to find the chair seat, the acquistion module (discussed below) would save the acquired chair seat samples on the instance list, which would then be available to the validation module for further processing.

To generate the subgoals of a goal, the elaborator checks first to see if any special modules exist for satisfying the goal for that particular object. This allows the user to provide advice to the planner about certain goals. For instance, the planner could be instructed to always look for the telephone on the tabletop. If no special routines are available, the planner looks for routines associated with the activity and that are generally useful for all objects.

For example, to elaborate the goal (ACQUIRE TABLE), the planner might be instructed to use a horizontal plane finder, since the tabletop is a horizontal plane. Alternatively, if there was no information associated with TABLE, the elaborator would look into general purpose routines pertinent to ACQUIRE. Figure 45 provides another example. There is a module for FIND that says to find an object, find its parts. Therefore, to find the chair a plan is generated for finding its major parts, the seat and the back.

In general, several planning modules may be associated with a given activity. In order to select the expansion modules appropriate to a particular goal, the planner evaluates a predicate function associated with each. If the predicate returns a true value, the planner adds the module to its list of applicable modules. Special cases of the predicate function can cause the module always to be selected (to provide defaults) or can cause the module to be selected by goals specified explicitly. This predicate allows us to have several modules, each an expert in a relatively narrow domain, and each selectable on the basis of that domain. For example, associated with the activity, ACQUIRE, could be a program for locating horizontal planes. This program would then have a predicate that checked that the object was horizontal.

Before a newly generated goal is added to the list of goals to be elaborated, a check is made to see if it has already been generated, or if any equivalent subgoals have been generated (the subgoal must have a special function to allow the planner to decide this). In either event, it is not processed further, and the subgoal list of the goal being expanded is made to point to the previous goal. The planner will also check for related subgoals (that can succeed or fail based on other goals that are not directly related, i.e., not parents or children). For example, the planner may generate a strategy that includes multiple filter subgoals where one filter predicate subsumes another. If the less restrictive predicate fails to turn up the desired object, it is

761

guaranteed that the more restrictive one also will not. Therefore, if the less restrictive filter is selected and executed, its outcome could imply the outcome of the related, restrictive filter. The planner maintains an explicit list of related subgoals, for this type of situation.

c. Scorer

After a plan has been elaborated, the next step is to decide which subgoal to attempt, or, in general, to decide what to do next. Obviously, the choice depends on the local expense of executing the goal, the more global expense due to the fact that some goals may be easier after other goals have been executed, and also the resulting likelihood of correct results after the goal has been executed.

We compute a score for a node which allows us to make a selection from proposed alternatives for satisfying the goal. This score function will be cost normalized by confidence, with the result that high confidence, low cost goals will have a smaller score than low confidence, high cost goals, and medium confidence, medium cost goals will fall somewhere in between.

Before discussing the tecnniques of scoring a plan, we digress briefly for a preview of plan execution. The plan executor receives a completely scored plan. It starts at the top node of the plan and proceeds downward through the plan, always selecting the best branch (on the basis of score) until it reaches a terminal node. This executable goal is then evaluated. If the subgoal fails, that failure is regarded as absolute, and the result is propagated backwards through the plan. However, if the node succeeds, then the confidence of the node is propagated back. That is, all nodes dependent on the successful node have their anticipated confidence altered to reflect the success of the node. When the propagation ceases, if the top goal has not been satisfied the updated plan is rescored and execution proceeds as above.

The output of the plan elaborator is a planning graph. This representation of the strategy, although usually nonreentrant, may, in general, have loops. While the problem of backing costs and confidence up through a tree structure is quite simple, the same problem in a graph with loops is not quite so straightforward. We will first describe the technique for a tree, and then show how it applies to a graph.

The scoring process uses a relaxation technique. All goals in the plan are put on a list to be examined. For each one in turn, the score is computed (if possible). If the score of a node changes from the last value, its parents are put on the list to have a new score calculated. The process continues until the scores stabilize. If a node has a score, it means that some way (presumably) exists to satisfy that goal.

Since the scores for terminal (executable) nodes do not change, the only thing we need consider for scoring purposes are the intermediate nodes. We begin the discussion by considering the way scores are backed-up from a subnode to its immediate predecessor.

There are four possible types of intermediate nodes, *AND, AND, OR, and ↑. The ↑ node merely passes its score upwards to its parents, and therefore need not be considered further. Since the *AND and AND nodes require all the subgoals to be satisfied, a score for this type of node can only be computed if scores for all the immediate subgoals have been computed. However, an OR node merely requires that one of the subgoals be satisfied. This means that even if only one of the subgoals can be scored, the OR node can be scored. Effectively, any unscored node is considered to have a score of infinity. The score of an AND or *AND node with an unscored subgoal is also infinite. Unscored subgoals of OR nodes can be ignored, and the score computed on the basis of the ones already scored. In practice, what occurs is that an initial score is determined for some subgoals, which allows other subgoals to receive scores. These subgoals in turn propagate their scores back, and the process continues to completion. At the end of the process, any nodes that have not received scores are deemed impossible to satisfy, and are deleted from the plan. In operation, the actual parameters that are backed up are cost and confidence, and the score of the node is computed from them directly.

We will use the symbol S to stand for score, C for confidence, and K for cost. We will also use T for the successful execution of a node, T^k for the sequence of successful executions of nodes 1 through k, $\overline{T}$ for unsuccessful execution of a node, and

$$\overline{T^k}$$

for the sequence of unsuccessful executions of nodes 1 through k. Thus, S_i is the score of the ith node, K_i the anticipated cost, and C_i the confidence. T_i is the representation of successful completion of node i.

The confidence is the a priori probability of a "good outcome" from the node. Obviously, the notion of a good outcome is directly related to the task the node is supposed to satisfy. For example, a good outcome from a filtering module would be that it discovered a few points on the target object. The a priori confidence requires that the node succeed for a good outcome to result, and is therefore the joint probability of a good outcome and success of the node. We make the standard Bayes' expansion to represent confidence as the product of two factors: the probablity of a good outcome from the node, given the success of the node, and the probability of success of the node. If we let Y_i represent a good outcome from the ith node, then we have

$$C_i = P(Y_i | T_i) P(T_i) .$$

We will also note that the a posteriori probability of a good outcome from a node is zero if it failed, and

$$P(Y_i|T_i)$$

if it succeeded.

As mentioned above, the score of a node will be the cost normalized by the confidence, that is

$$S = K/C$$

for the score function. In the course of searching for the best path through the graph, we will attempt to minimize this function at each node. In order to compute the score of intermediate nodes, we will back-up costs and confidences from subnodes to their superiors. We will discuss techniques for that now.

i) Scoring an OR Node

We begin the discussion with the OR node in Figure 46. We assume that the subnodes are ordered by increasing score. The backed-up parameters will be dependent on the order, and it can be shown that this particular ordering will result in a minimum score for the node.

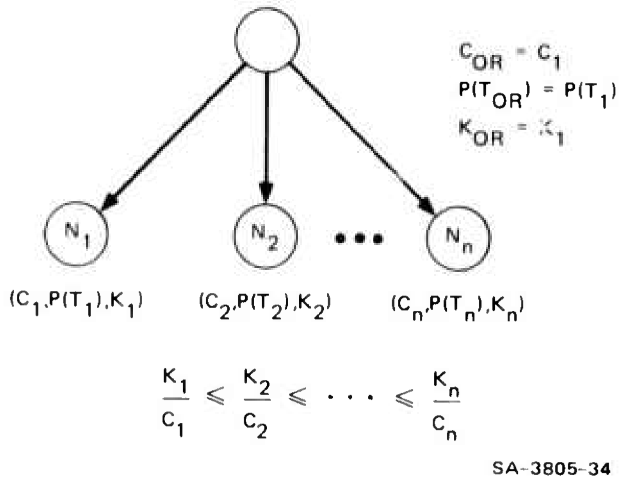


FIGURE 46 BACKED-UP PLANNING PARAMETERS FOR AN OR NODE

For an OR node to have a good outcome, one of the subnodes must have a good outcome. Therefore, we can write the confidence as the sum of the probabilities of a good outcome from each sequential node. That is,

$$C_{OR} = P(Y_i, T_i) + P(Y_2, T_2, \overline{T_1}) + \dots + P(Y_i, T_i, \overline{T^{i-1}}) + \dots + P(Y_n, T_n, \overline{T^{n-1}}) \qquad (D1)$$

or,

$$C_{OR} = \sum_{i=1}^{n} P(Y_i, T_i, \overline{T^{i-1}}) \qquad (D2)$$

Expanding this formula gives

$$C_{OR} = \sum_{i=1}^{n} P(\gamma_i | T_i, \overline{T^{i-1}}) P(T_i | \overline{T^{i-1}}) P(\overline{T^{i-1}}) \qquad (D3)$$

We have in

$$P(\gamma_i, T_i, \overline{T^{i-1}})$$

a context sensitive measure of confidence. That is, it is a confidence measure that takes the past history of the parent node into account. As this will, in general, be difficult to compute, we will make the weakening assumption of independence between T_i and T^{i-1}, and therefore

$$\overline{T^{i-1}}$$

We also assume that γ_i is dependent only on T_i, not on

$$\overline{T^{i-1}}.$$

With this assumption, we can rewrite the formula as

$$C_{OR} = \sum_{i=1}^{M} P(\gamma_i, T_i) P(\overline{T^{i-1}}) \qquad (D4)$$

or

$$C_{OR} = \sum_{i-1}^{M} C_i \, P(\overline{T^{i-1}}) \qquad (D5)$$

And finally we have

$$C_{OR} = \sum_{i=1}^{n} C_i \prod_{j=1}^{i-1} [1 - P(T_j)] \qquad (D6)$$

The expected cost of the node is the sum of the expected costs of each of the individual subnodes, which is just the cost of the node times the probability of executing it. The probability of executing a node is the probability that all preceding nodes fail. Therefore, we can write the expected cost of the node as

$$K_{OR} = \sum_{i-1}^{n} K_i \, P(\overline{T^{i-1}}) \qquad (D7)$$

With the same assumptions of independence as above, we can write the cost formula as

$$K_{OR} = \sum_{i=1}^{n} K_i \prod_{j=1}^{i-1} [1 - P(T_j)] \qquad (D8)$$

The probability of success of the OR node is just

$$1 - P(\overline{T^n}),$$

that is, the probability that they do not all fail. This is rewritten

$$P(T_{OR}) = 1 - \prod_{i=1}^{n} [1 - P(T_i)] \qquad (D9)$$

The score is computed from the backed-up cost and confidence. It is apparent, however, that a very expensive subnode with low reliability would pull the score of the whole node down, even though it might be likely that it would never be executed. Since we would like the score of a node to reflect the best that we can expect from the node, we will define the score to be the minimum score of the set of subsequences of the sequence of subgoals. That is, we can compute the score for the whole sequence, then for the subsequence without the last goal, then without the last two subgoals, and so on. We then take the minimum of these scores to be the score of the node. But this definition implies that the score of an OR node will always be the score of the first subgoal in the sequence. Therefore, the first subgoal in the sequence will represent the node. As a result, the planning parameters for the node become:

$$C_{OR} = C_1$$

$$K_{OR} = K_1$$

$$P(T_{OR}) = P(T_1)$$

$$S_{OR} = S_1$$

It is important to note that this result is strictly due to the choice of score function. The more general results will be required in later work with different score functions.

ii) Scoring an AND Node

We assume that the subnodes of an AND node have been ordered by

$$K_i/(1 - C_i),$$

as this can be shown to result in a minimum score for the node. A *AND node is already ordered, so the formulas derived for an AND node will apply equally to a *AND node. The AND node is shown in Figure 47.

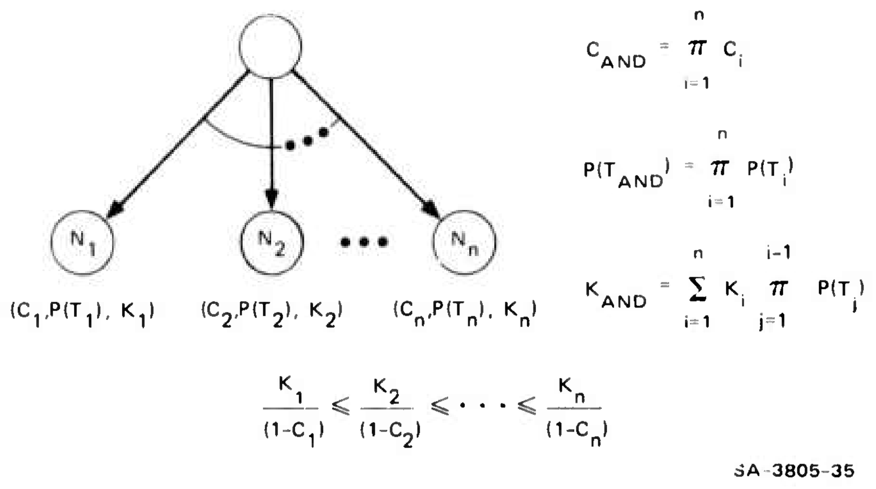

N_1
N_2
N_n
$(C_1, P(T_1), K_1)$
$(C_2, P(T_2), K_2)$
$(C_n, P(T_n), K_n)$
$C_{AND} = \prod_{i=1}^{n} C_i$
$P(T_{AND}) = \prod_{i=1}^{n} P(T_i)$
$K_{AND} = \sum_{i=1}^{n} K_i \prod_{j=1}^{i-1} P(T_j)$
$\frac{K_1}{(1-C_1)} \leqslant \frac{K_2}{(1-C_2)} \leqslant \cdots \leqslant \frac{K_n}{(1-C_n)}$
SA-3805-35

FIGURE 47 BACKED-UP PLANNING PARAMETERS FOR **AND** AND ***AND** NODES

The confidence in this case is the probability that all the subgoals will have a good outcome, or

$$C_{AND} = P(Y^n, T^n) = P(Y^n | T^n) P(T^n)$$

with our assumptions of independence, we have

$$P(Y^n | T^n) = \prod_{i=1}^{n} P(Y_i | T_i) \qquad (D10)$$

and

$$P(T^n) = \prod_{i=1}^{n} P(T_i) \qquad (D11)$$

Therefore,

$$C_{AND} = \left[\prod_{i=1}^{n} P(\gamma_i | T_i)\right]\left[\prod_{i=1}^{n} P(T_i)\right] \qquad (D12)$$

or

$$C_{AND} = \prod_{i=1}^{n} P(\gamma_i | T_i)\ p(T_i) \qquad (D13)$$

And, therefore,

$$C_{AND} = \prod_{i=1}^{n} C_i \qquad (D14)$$

If a subnode of an AND node fails, we terminate execution. Therefore, the expected cost of a single subnode in the sequence is the cost of the node times the probability of executing it. This probability is the probability of all preceding nodes succeeding. The cost, therefore, is

$$K_{AND} = \sum_{i-1}^{n} K_i\ P(T^{i-1}) \qquad (D15)$$

or,

$$K_{AND} = \sum_{i=1}^{n} K_i \prod_{j=1}^{i-1} P(T_j) \qquad (D16)$$

The probability of success of the node is

$$P(T_{AND}) = \prod_{i=1}^{n} P(T_i) \qquad (D17)$$

The new score is computed from these backed-up values.

iii) Scoring a Plan

With these techniques for computing the planning parameters of individual nodes, organizing the planning tree becomes a straightforward task. The parameters of the terminal nodes are backed-up to their predecessors. These intermediate nodes, in turn, pass their parameters on to their parents. Finally, the top node receives a score and the process is complete.

However, the process of scoring a plan with loops (such as the one in Figure 45) is more complicated.

The existence of a loop in the plan means that some node depends on the solution of another node for its solution, but the second node also depends on the first for its solution.

Obviously, if these nodes had no other alternatives for their solution, there would be no solution; but, the situation is usually that shown in Figure 48. (Notice that this graph could be a subgraph of the one shown in Figure 45). Here, node 7 has node 5 as an option for its solution. Node 5 requires node 8 which has node 6 as one of its options, but node 6 requires node 7. Obviously, either node 1 or node 3 must be executed first. If we assume node 1 is executed first and succeeds, the choice remains whether to then execute node 3 or node 4, since success for either of these nodes means success of the plan.

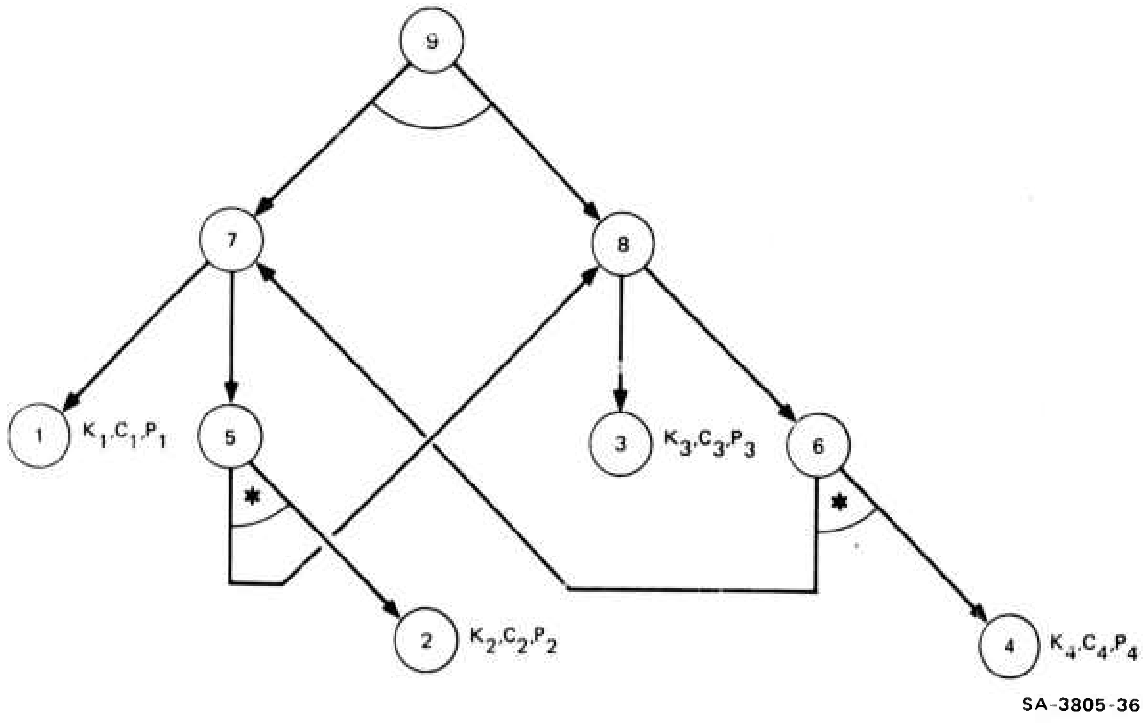


FIGURE 48 PLANNING GRAPH WITH LOOPS

The program that scores a plan assumes that the initial score of a nonterminal node is infinite. An AND (or *AND or ↑) node can receive a noninfinite score only if all of its subgoals have noninfinite scores. An OR node can receive a noninfinite score if any of its subgoals has a noninfinite score. Therefore, in the graph, nodes 7 and 8 can receive initial scores of

$$K_1/C_1$$

and

$$K_3/C_3$$

respectively. Using the parameters from node 7, node 6 can be scored. If the score for node 6 is less than

$$K_3/C_3,$$

then node 8 receives a new score. In any case, node 5 can also receive a score since node 8 now has one. Again, if the score of node 5 is less than that for node 1, then node 7 receives a new score, node 6 receives a new score, and node 8 receives a new score. It is important to notice that if the score of node 5 is less than that for node 1, and this is reflected back to node 6, it is impossible for the new score of node 6 to be less than that for node 3. If it were less, it would mean that costs around the loop decreased and/or confidence increased, which is not possible. Therefore, either the final score of node 5 will be greater than that of node 1 or the final score of node 6 will be greater than that of node 3. We also see that there will be a maximum of two iterations around the loop before things settle down.

To be more precise, we will perform the computations with symbolic quantities for the parameters of terminal nodes 1, 2, 3, and 4. S_7 the initial score for node 7, is

$$K_1/C_1 .$$

From this, we can compute

$$S_6 = \frac{K_1 + P_1K_4}{C_1C_4} \tag{D18}$$

and, therefore,

$$S_8 = \min\left\{K_3/C_3, \frac{K_1 + P_1K_4}{C_1C_4}\right\} . \tag{D19}$$

S_5 is either

$$\frac{K_3 + P_3K_2}{C_3C_2}$$

or

$$\frac{K_1 + P_1 K_4 + P_1P_4K_2}{C_1C_4C_2},$$

depending on which was the minimum. This means that the revised score for node 7 is either

$$\min\left\{K_1/C_1, \frac{K_3 + P_3K_2}{C_3C_2}\right\}$$

or

$$\min\left\{K_1/C_1, \frac{K_1 + P_1K_4 + P_1P_4K_2}{C_1C_4C_2}\right\}.$$

But the latter minimum is

$$K_1/C_1,$$

since

$$\frac{K+A}{CB} \geq \frac{K}{C} \tag{D20}$$

for K and A greater than 0 and C and B between 0 and 1. Therefore,

$$S_7 = \min\left\{ K_1/C_1, \frac{K_3 + P_3K_2}{C_3C_2} \right\}, \tag{D21}$$

since if

$$\frac{K_3 + P_3K_2}{C_3C_2} < \frac{K_1}{C_1} \tag{D22}$$

$$S_8 = K_3/C_3$$

$$S_s = \frac{K_3 + P_3K_2}{C_3C_2}. \tag{D23}$$

The final result is

$$S_8 = \min\left\{ K_3/C_3, \frac{K_1 + P_1K_4}{C_1C_4} \right\} \tag{D24}$$

and

$$S_7 = \min\left\{K_i/C_1, \frac{K_3 + P_3K_2}{C_3C_2}\right\}. \qquad (D25)$$

Now, if

$$K_1/C_1 < K_3/C_3$$

then

$$S_7 = K_1/C_1$$

and

$$S_8 = \min\left\{K_3/C_3, \frac{K_1 + P_1K_4}{C_1C_4}\right\}. \qquad (D26)$$

What this means is that the first node to execute is 1, but the next depends on the relative magnitudes of

$$K_3/C_3$$

and

$$\frac{K_1 + P_1 K_4}{C_1 C_4}.$$

d. Execution

After plan scoring is complete, the execution program selects the best executable subgoal, and evaluates it. The success or failure is propagated upwards to all parents, and any related subgoals are dealt with also. If the strategy has not succeeded or failed at this point, the planner calls the scorer to rescore the net and then continues with the next goal.

To do this, the executor starts at the top goal of the net, and proceeds downwards through the subgoals. Wherever there is an option, it chooses the subgoal with the lowest score. When it finds a node where the best subgoal is executable, it passes that subgoal to an evaluation function. This function executes the subgoal and propagates the success or failure backwards. It keeps track of subgoals that were executed, along with their outcomes on a special list.

The propagation of failure is straightforward. If any subgoal of an AND or *AND goal fails, then the goal itself also fails and continues the propagation upward. A subgoal of an OR node that fails is merely removed from the list of subgoals. When any subgoal list is reduced to a single subgoal, the operator is replaced with the ↑ operator which always propagates the outcome of the subgoal directly upward.

When a subgoal succeeds, its a priori confidence value, C, is replaced by the a posteriori confidence,

$$C/P(T) ,$$

that is, the probability of a good outcome given that the node succeeded. The cost is set to zero, and P(T) to one. This allows the parameters of the predecessor nodes to be adjusted to reflect the success. If the first subgoal of an AND node succeeds, then the confidence of the node becomes

$$\prod_{i=1}^{n} C_i$$

After the kth node succeeds, the confidence is

$$\prod_{i=1}^{n} C_i / \prod_{i=1}^{k} P(T_i)$$

The remaining cost after success of the kth node is

$$\sum_{i=k+1}^{n} K_i \prod_{j=k+1}^{i-1} P(T_j)$$

Since the confidence is incr sing, and the cost decreasing, the score of the node decreases. This means that the subgoals of the node will be executed in order as long as they are successful. The final confidence of a successful AND node is

$$\prod_{i=1}^{n} C_i / \prod_{i=1}^{n} P(T_i)$$

If a subgoal of an OR node succeeds, then the OR node itself succeeds with a posteriori confidence,

$$C/P(T).$$

This is propagated back along with the cost (which is zero) and P(T) (which is one) to the parents of the OR node.

After propagating the outcome of a goal, the executor checks the list of related goals to see if any other action needs to be taken. Since in many cases the outcome of one goal can determine the outcome of another, this is the point at which action is taken.

If the outcome of a subgoal does not propagate to the top goal, the net has a new score computed for it in exactly the same way as it did initially, and the execution continues with the next goal. If, at some point, a graph with a loop is left, and there are no terminal nodes that will allow for a solution, then no failure will propagate to the top. Instead, the top goal will not receive a score. This is considered a failure of the top goal.

e. Conclusions

We have described an artificial intelligence planning and execution program that is able to make cost effective decisions about what to do next in analysing scenes. We have described in some detail the effects of a particular score function (i.e., K/C . This is probably not the best possible score function. Another possibility that will be explored is to treat an OR node as if there were a discrete cost-confidence function associated with it. That is, considering each subgoal, there is a certain reliability to be expected for a certain cost. We can consider these cost versus reliability values as points on a curve, and the complete set provides a cost-confidence function. This function can then be propagated upward instead of making a decision about what the score should be at the local node level. This way, the parameters can be passed up to the executor which can then decide (using a variety of criteria) just what should be done. With this kind of information available, the executor could use budgetary considerations (how much resource is available to expend for the job) or considerations about required reliability.

3. MSYS: A System for Reasoning about Scenes*

a. Introduction

This section describes MSYS, an operational system that interprets scenes. Scene interpretation is formulated as a constraint optimization problem. Given a set of regions from a partitioned scene, a set of possible interpretations for each region derived from local surface attributes (e.g., color, orientation), and a set of constraints on spatial relationships of interpretations, MSYS deduces the most probable global interpretation for each region. The system has been used to interpret manually partitioned room scenes using a variety of semantic constraints.

----

*The work reported in this section was partially supported by ONR under Contract No. N0014-71-C-0294 [58].

Scene interpretation has been previously approached both as an exercise in deductive reasoning and as a problem in heuristic optimization. As a reasoning exercise, the objective was to deduce unique local interpretations for each region that were consistent with symbolic global constraints [59-61]. As an optimization problem, the objective was to find the set of local interpretations with the highest joint likelihood, where each interpretation likelihood was a function both of the local region attributes and the likelihoods of semantically related interpretations assigned to other regions [62-64].

The most successful deductive scene interpretation system was created by Waltz [61] for analyzing ideal line drawings of scenes containing toy blocks. Lines were labeled at each vertex with a set of possible interpretations, such as concave, convex, shadow, and crack. These initial interpretations were based on a catalog listing the various ways legal vertices of trihedral solids could appear in a line drawing. Some line interpretations could be immediately rejected as impossible if they were not assigned at both ends of the line. Eliminating a line interpretation would eliminate one or more possible interpretations for each of its vertices, which in turn, could eliminate additional interpretations of other lines joining at those vertices. This elimination process would often propagate until each line was left with a unique interpretation.

The deductive approach to scene analysis is limited to symbolic problem domains such as line drawing analysis, in which all interpretations that violate constraints are dismissed absolutely and all those that do not are considered equally likely. Most real perception problems, by contrast, entail noisy measurements and probabilistic constraints. Thus, it is not meaningful to speak of correct or incorrect interpretations, but only of the most probable interpretations, given current evidence. A number of investigators have therefore chosen to view scene interpretation as a heuristic optimization problem.

In scene interpretation, the likelihood of each region interpretation theoretically depends, at least indirectly, on the likelihood of every other region interpretation. Because of this interaction, the search space explodes exponentially with both the number of regions and the universe of possible interpretations. Numerous simplifying assumptions have been proposed to make the search tractable. Both Duda [59] and Guzman [60] performed a limited tree search using local binary valued constraints on legal adjacency; interpretations were assigned to regions in order of maximum local likelihood, subject to semantic consistency with interpretations previously selected for adjacent regions. Eliminating all possible interpretations for an unassigned region forced reconsideration of an earlier assignment. Yakimovsky developed two algorithms which, while still based on tree search, utilized wider classes of constraints [63]. Algorithm A allowed real valued boundary constraints which could alter the local likelihood of an interpretation based on the attributes and interpretation likelihoods of adjacent regions. These constraints were used following an instantiation to update the interpretation likelihoods of uninstantiated regions immediately adjacent to the newly instantiated one. A branch and bound condition on the product of all instantiated interpretations was used to terminate search along unpromising lines. Algorithm B allowed real valued constraints between arbitrary pairs of region interpretations. These constraints were again used only to update interpretation likelihoods of uninstantiated regions, that were directly related to a newly instantiated interpretation. The use of real valued constraints provided Yakimovsky with a more sensitive measure of global likelihood for assigning interpretations to regions. However, since both his algorithms limited constraint interaction (by freezing the likelihoods of instantiated interpretations) in the interests of search efficiency, the resulting likelihoods represent only an approximate simultaneous solution of the constraints.

Besides obtaining only approximate solutions, all of the above methods share another inherent fault; hard won information is frequently thrown away when the search backtracks. In one of Duda's examples, a particular region is deduced not to be an electrical outlet on at least five separate branches of the interpretation tree. In each context, outlet was excluded as a possible interpretation because the region in question was adjacent to a region that had already been postulated to be the floor. It was precisely this type of redundant reasoning that Waltz sought to avoid, by eliminating inconsistent interpretation before embarking on search. It occurred to us that a similar deductive scheme might be used with real valued constraints to depress the likelihood of inconsistent interpretations. An interpretation with low enough likelihood would effectively be eliminated because it would never be selected for instantiation. The consequent reduction in wasted search effort could then be redirected toward a more optimal global determination of interpretation likelihoods.

The following section describes a realization of these objectives.

b. M* - An Optimization Algorithm Using Constraint Propagation

This section outlines an algorithm for scene interpretation that combines the best features of deduction and optimization [58]. Each region in a partitioned scene is assigned all interpretations permitted by its local attributes (e.g., it may be door or wall on the basis of color, size, and surface orientation). A likelihood is associated with each interpretation, based initially on the a priori likelihoods of alternative interpretations for the region. These local likelihoods are then reevaluated based on the current likelihoods of all semantically related interpretations. This simultaneous reevaluation is accomplished on a serial computer by a relaxation process that culminates with a set of global equilibrium likelihoods. This relaxation process is the analog of Waltz's deduction process for real valued constraints and, like that process, can be implemented using a propagation technique, as will be discussed in Subsection c below.

The resulting equilibrium likelihoods represent estimates of the global validity of alternative interpretations in the current context. If all regions are left with a single probable interpretation, the analysis terminates successfully. On the other hand, if all interpretations of any regions are judged unlikely, the analysis is abandoned on grounds that the context is inconsistent. The most usual case finds at least one region still left with several possible interpretations. The analysis then reverts to a search for the set of region interpretations having the highest joint likelihood.

The search proceeds at each step by restoring the highest scoring context (initially the global context) and instantiating the most likely ambiguous interpretation remaining in that context. Equilibrium likelihoods are recomputed, based on the new instantiation, and used, as in the global context, to decide the future course of the search. If all regions now have unique interpretations, the search is terminated, if any region has no possible interpretations, the context is abandoned, otherwise, a global score for the context is computed by summing the equilibrium likelihoods of the currently best interpretation for each region. The search then continues in the currently best context.

In Ref. 58 it is proved that this algorithm, with reasonable restrictions on the nature of the constraints, will terminate with the optimal set of interpretations for those constraints. The resulting equilibrium likelihoods will, in general, be better estimates of actual global likelihoods than those obtained by the methods described in Subsection a above. Moreover, it is shown that, in finding the optimum, this new search will instantiate no more interpretations than any of the conventional tree searching methods. The improved estimates of interpretation likelihood provided by the global relaxation process improve the order of instantiation and context selection at each stage of search, thereby minimizing backtracking. The relaxation process can thus be considered a new type of global lookahead embedded in a conventional best first search algorithm. All of the algorithms discussed in Subsection a are in fact special cases of our algorithm utilizing limited propagation and/or constraints [58].

The number of instantiations is, of course, only one measure of search efficiency. The computational effort expended in lookahead must also be considered in assessing which of two optimization approaches is more efficient overall. The relaxation procedure is, admittedly, a computationally expensive form of lookahead, but it can be very cost-effective in problem domains with large numbers of hypotheses interacting through many pairwise constraints. This cost-effectiveness stems from the fact that many inconsistencies are deduced once and for all in the global context rather than having to be repeatedly rediscovered for individual instantiations as in conventional backtrack searches.

c. MSYS - An Implementation of M*

Implementation of the M* optimization algorithm described above requires a relaxation mechanism for constraint evaluation and a context switching mechanism for search. We will describe generalized implementations for these two components, which are designed to facilitate experimentation with a variety of knowledge representations, search strategies, and control structures (including specifically all strategies mentioned in Subsection a).

1) Constraint Evaluation--The XDEMON System

The simultaneous determination of equilibrium likelihoods for a mutually constrained set of interpretations, inherently a parallel operation, has been

----

*A detailed implementation of MSYS using these components appears in Ref. 58.

efficiently implemented on a serial computer by simulating a network of asynchronous parallel processes interacting through a global data base. The data base consists of variables representing constrained entities and constraints. Associated with each variable is a procedure for computing a value in terms of the current values of other variables. Each variable also has a list of related variables whose procedures utilize the present variable as input. When the cumulative change in the value of a variable exceeds a threshold, its related variables are activated by adding their procedures to a stack of jobs to be run. Thus, if running a process changes the value of its associated variable, additional processes may be activated. Execution terminates when the job stack is empty.

The evaluation process is initiated by loading the job stack with the procedures of variables for which updated global likelihoods are desired. Initially, the procedures of every variable are put on the job stack to obtain global equilibrium likelihoods. The consequences of subsequent instantiations are explored by queueing only the procedures of variables directly dependent on the values of the instantiated variables.

The above scheme for establishing global likelihoods is applicable to a variety of constraint satisfaction and optimization problems. A set of LISP functions, known collectively as the system XDEMON, has been developed to facilitate the creation of network representations for particular constraint problems. These functions were documented in Appendix B of Ref. 37. For our scene interpretation application, variables are used to represent regions, region interpretations, and relational constraints between pairs of regions (e.g., adjacent, above, and the like). Region variables evaluate to boundary descriptions, interpretation variables evaluate to the current global likelihood of that interpretation, and constraint variables evaluate to the degree to which argument regions conform to the stipulated constraint.

The procedure associated with each interpretation variable computes current likelihood as a function of local likelihood (i.e., a likelihood based solely on a region's own attributes) and global likelihood (i.e., a likelihood based on the current likelihoods of semantically related interpretations and on the values of corresponding constraint variables expressing how well each relation is satisfied). The variables are interconnected so that the reevaluation of any region variable (reflecting a change in boundary) will cause all contextual constraints involving that region to be reevaluated. Similarly the reevaluation of any interpretation or constraint variables will automatically queue all semantically related interpretations for reexamination. The efficiency of the scheme stems from the fact that variables are only reevaluated when the value of a contextually related variable changes.

ii) Search

A simple state saving mechanism has been programmed that allows a current computational context to be reinstated at a future time. The search context for A* consists of the complete network of variables described in Subsection b above plus additional problem dependent variables that typically may include a global score, a list of previous region instantiations, and a priority queue (called IQUEUE) of instantiations yet to be tried in that context. The saved context is inserted onto a priority queue of contexts (SQUEUE) to be explored. In general, a search proceeds by reinstating the top priority context on SQUEUE, selecting the best instantiation from the current IQUEUE, and then reevaluating the network of variables in the new context created by that instantiation. An acceptable solution terminates the search. Otherwise, IQUEUE is updated and the new context added to SQUEUE. The search then continues in the top context.

The nature of the search will be determined by the problem dependent priority functions used to update SQUEUE and IQUEUE and by the termination condition. A depth first search is obtained by always adding new contexts to the top of SQUEUE, a breadth first search is had by always adding them to the bottom, and a best first search is realized by queueing contexts in order of global score [65]. Heuristic guidance is introduced into the search by the function that updates IQUEUE. The termination condition can be chosen to select the highest scoring solution (i.e., a complete set of globally consistent instantiations), the first solution encountered, or a complete enumeration of all solutions.

d. Using MSYS

An interpretation problem is posed to MSYS in the following way. First, the experimenter, using a trackball, circles a set of test regions on the displayed image of a scene, as indicated in Figure 49. These regions are retained on a disk file. Next he enters the constraints to be used in the current experiment. He may also directly assert symbolic relationships between regions (e.g., that two regions be considered adjacent). This ability was useful for simulating unimplemented relational procedures and for fabricating experimental situations. Interpretation is initiated by calling the function Interpret with a region file name or a list of regions as an argument. MSYS responds with a complete protocol of the interpretation process containing, first, a list of locally possible region interpretations and their initial likelihoods, second, a trace of all jobs executed from the job list and, third, a final list of unique region interpretations with associated likelihoods (or else a message announcing failure to find consistent interpretations).

Constraints are entered in the format (ADDCONST Rel Vlist). Rel is a simple relation such as (Above Chairback Chairseat), a boolean expression of simple relations such

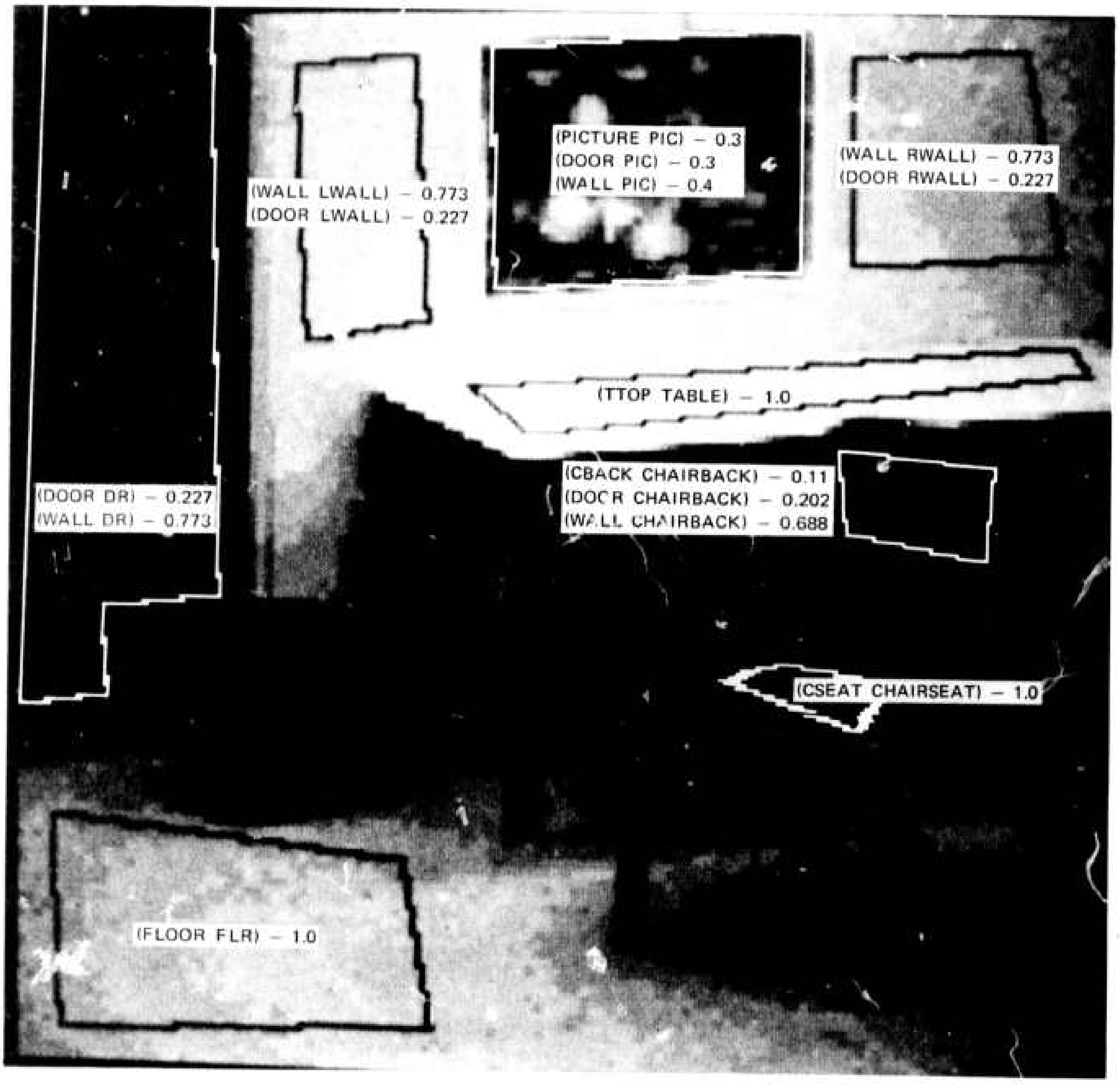


FIGURE 49 MANUALLY PARTITIONED ROOM SCENE WITH LOCAL BAYESIAN INTERPRETATION LIKELIHOODS BASED ON HEIGHT AND SURFACE ORIENTATION

as (Or (Adjacent Picture Wall) (Adjacent Picture Frame)), or a functional constraint such as (Homogeneous Door). Independent constraints on the same interpretation are assumed to be embedded in an implicit conjunction. Vlist is a list of the interpretations to which a constraint applies. Thus (ADDCONST (Adjacent Picture wall)) (Picture)) requires all Pictures to be adjacent to Walls but puts no constraint on Walls. If Vlist is omitted, MSYS assumes that the constraint applies reciprocally to all interpretations mentioned within it.

e. An Example

The operation of MSYS will be illustrated by describing an interpretation of the scene partition shown in Figure 49, using the constraints given in Figure 50. These particular constraints were empirically selected for this room scene domain on the basis of their computational simplicity and their suitability for use in partially partitioned scenes. Alternative constraints come

```
[ADDCONST (QUOTE (NOT* (ADJ DOOR PICTURE]
(ADDCONST (QUOTE (FUNCTION ROOMPART))
          (QUOTE (DOOR WALL)))
(ADDCONST (QUOTE (FUNCTION HOMO))
          (QUOTE DOOR))
(ADDCONST (QUOTE (FUNCTION HOMO))
          (QUOTE WALL))
(ADDCONST (QUOTE (FUNCTION HOMO))
          (QUOTE CBACK))
```

FIGURE 50 CONSTRAINTS FOR ROOM SCENE INTERPRETATION

readily to mind: (ADJACENT PICTURE WALL) and (NOT (ABOVE DOOR WALL)), to name two. No formal basis yet exists for choosing particular constraints.

The first step in analysis is the assignment of possible interpretations to the regions based on their local attributes. Figure 51 shows the interpretation variables created during this initial labeling phase and their associated local likelihoods. These local interpretations were obtained using a Bayesian classifier, which compared the height and surface orientation of regions in Figure 49 with those of training regions previously outlined in a similar scene.* For each region only those interpretations with a likelihood exceeding ten percent of the likelihood of the most probable interpretation were retained. Initial classification was based on height and orientation, because these were considered more intrinsic attributes than color and brightness. The homogeneity of color and brightness over a given surface, however, provided a key global constraint. Note that each horizontal surface received a unique interpretation determined by its height, but that all vertical surfaces received at least two possible

----

*Height and orientation are obtained directly from range data using transformations described in Ref. 66. The data used in our experiments simulated the output of a developmental time of flight laser range finder [37], whose current accuracy is about one inch in ten feet.

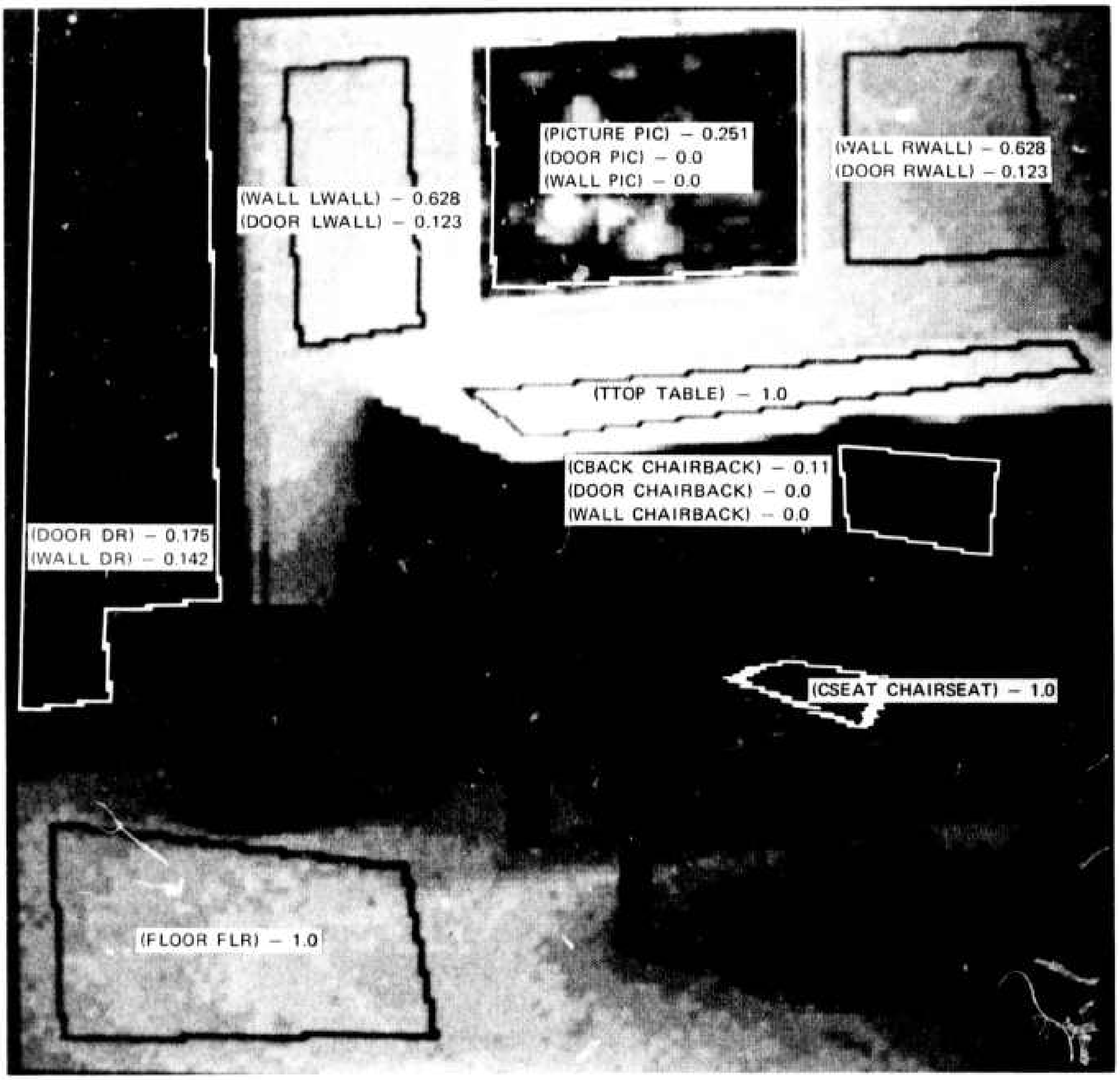


FIGURE 51 INITIAL EQUILIBRIUM LIKELIHOODS BEFORE SEARCH

interpretations (door, wall, and when consistent with height extremes, wastebasket, chairback, or picture). Global constraints were thus only needed for vertical surfaces. The Bayesian likelihoods associated with the interpretations of vertical regions represent the expected proportion of the image occupied by each interpretation. The likelihoods assigned to (Wall Dr) and (Door Dr) in Figure 51, for example, represent the relative areas occupied by Wall and Door in the test image.

The second step of analysis entails the construction of evaluation functions for each interpretation variable and the subsequent computation of global equilibrium likelihoods. The state of the data base following the attainment of equilibrium is shown in Figure 52. (For conciseness, the uniquely interpreted horizontal regions have been omitted.)

The evaluation function for each interpretation is formulated as a conjunction of local likelihood (see Figure 52) and support for applicable constraints. The support for each constraint is expressed as a logical combination of other interpretation likelihoods and region relations, indicating all the possible ways that the constraint can be satisfied in the current image. The actual process of creating these evaluation functions is described in Ref. 58.

```
VARIABLE: (DOOR RWALL)
VALUE:      .123
PROCEDURE:
            [AND* .227 (AND* (NOT* (OR* (DOOR DR)
                                        (DOOR PIC)
                                        (DOOR CHAIRBACK)))
                             (NOT* (AND* (ADJ RWALL PIC)
                                         (PICTURE PIC]
RELATIVES:
            ((OPTION (DOOR RWALL) (WALL RWALL)))
            ((AND* (ADJ RWALL PIC) (DOOR RWALL)))
            ((OR* (DOOR LWALL) (DOOR RWALL)))


VARIABLE: (WALL RWALL)
VALUE:      .628
PROCEDURE:
            [AND* .773 (NOT* (OR* (WALL DR)
                                  (WALL PIC)
                                  (WALL CHAIRBACK]
RELATIVES:
            ((OPTION (DOOR RWALL) (WALL RWALL)))
            ((OR* (WALL LWALL) (WALL RWALL)))


VARIABLE: (DOOR LWALL)
VALUE:      .123
PROCEDURE:
            [AND* .227 (AND* (NOT* (OR* (DOOR DR)
                                        (DOOR PIC)
                                        (DOOR CHAIRBACK)))
                             (NOT* (AND* (ADJ LWALL PIC)
                                         (PICTURE PIC]
RELATIVES:
            ((OPTION (DOOR LWALL) (WALL LWALL)))
            ((AND* (ADJ LWALL PIC) (DOOR LWALL)))
            ((OR* (DOOR LWALL) (DOOR RWALL)))


VARIABLE: (WALL LWALL)
VALUE:      .628
PROCEDURE:
            [AND* .773 (NOT* (OR* (WALL DR)
                                  (WALL PIC)
                                  (WALL CHAIRBACK]
RELATIVES:
            ((OPTION (DOOR LWALL) (WALL LWALL)))
            ((OR* (WALL LWALL) (WALL RWALL)))
```

FIGURE 52   STATE OF DATA BASE BEFORE SEARCH

```
VARIABLE: (DOOR CHAIRBACK)
VALUE:     0.0
PROCEDURE:
           (DOOR CHAIRBACK)
RELATIVES:
           ((OPTION (DOOR CHAIRBACK) (CBACK CHAIRBACK) (WALL CHAIRBACK)))
           ((OR* (DOOR DR) (DOOR PIC) (DOOR CHAIRBACK)))


VARIABLE: (CBACK CHAIRBACK)
VALUE:     .11
PROCEDURE:
           (CBACK CHAIRBACK)
RELATIVES:
           ((OPTION (DOOR CHAIRBACK) (CBACK CHAIRBACK) (WALL CHAIRBACK)))


VARIABLE: (WALL CHAIRBACK)
VALUE:     0.0
PROCEDURE:
           (WALL CHAIRBACK)
RELATIVES:
           ((OPTION (DOOR CHAIRBACK) (CBACK CHAIRBACK) (WALL CHAIRBACK)))
           ((OR* (WALL DR) (WALL PIC) (WALL CHAIRBACK)))


VARIABLE: (PICTURE PIC)
VALUE:     .251
PROCEDURE:
           (AND* .3 (NOT* (OR* (AND* (ADJ RWALL PIC)
                                     (DOOR RWALL))
                               (AND* (ADJ LWALL PIC)
                                     (DOOR LWALL)
RELATIVES:
           ((OPTION (PICTURE PIC) (DOOR PIC) (WALL PIC)))
           ((AND* (ADJ RWALL PIC) (PICTURE PIC)))
           ((AND* (ADJ LWALL PIC) (PICTURE PIC)))


VARIABLE: (DOOR PIC)
VALUE:     0.0
PROCEDURE:
           (DOOR PIC)
RELATIVES:
           ((OPTION (PICTURE PIC) (DOOR PIC) (WALL PIC)))
           ((OR* (DOOR DR) (DOOR PIC) (DOOR CHAIRBACK)))


VARIABLE: (WALL PIC)
VALUE:     0.0
PROCEDURE:
           (WALL PIC)
RELATIVES:
           ((OPTION (PICTURE PIC) (DOOR PIC) (WALL PIC)))
           ((OR* (WALL DR) (WALL PIC) (WALL CHAIRBACK)))
```

FIGURE 52 STATE OF DATA BASE BEFORE SEARCH (Continued)

301

```
VARIABLE: (DOOR DR)
VALUE:      .175
PROCEDURE:
            (AND* .227 (AND* (NOT* (OR* (DOOR LWALL)
                                        (DOOR RWALL)))
                             (NOT* 0.0)))
RELATIVES:
            ((OPTION (DOOR DR) (WALL DR)))
            ((OR* (DOOR DR) (DOOR PIC) (DOOR CHAIRBACK)))


VARIABLE: (WALL DR)
VALUE:      .142
PROCEDURE:
            (AND* .773 (NOT* (OR* (WALL LWALL)
                                  (WALL RWALL)]
RELATIVES:
            ((OPTION (DOOR DR) (WALL DR)))
            ((OR* (WALL DR) (WALL PIC) (WALL CHAIRBACK)))
```

FIGURE 52   STATE OF DATA BASE BEFORE SEARCH (Concluded)

The constraint (Homogeneous Wall), requires that all regions interpreted as wall have similar brightness. This constraint introduces a clause of the form (NOT* (OR* (WALL Ri1)---(Wall Rin))) into the likelihood function of each wall interpretation (Wall Ri). The disjunction includes all regions admitting the interpretation wall, whose brightness differs from that of Ri by more than ten percent. The effect is to reduce the likelihood that region Ri is wall by an amount proportional to the likelihood that these other, nonhomogeneous regions are thought to be wall. Homogeneity constraints on other interpretations have analogous effects.

The constraint, Roompartition, requires that the brightness of all regions admitting the interpretations Wall or Door (i.e., surfaces that partition rooms) be similar to that found at the very top of the image vertically above their centers of gravity. Region interpretations failing this test are rejected outright by pinning their likelihood to zero. The likelihood of interpretations that pass is unaffected. The Roompartition constraint is closely related to the previously discussed homogenerity constraint and was included to handle cases where that constraint was ineffectual because the experimenter failed to circle enough regions. It is based on an assumption that, in a standard view of a room scene, wall or door will appear at the top of the image, and moreover, that walls and doors never appear vertically above each other. The effect of this constraint was to eliminate

"Door" and "Wall" as possible interpretations of vertically oriented regions lacking appropriate vertical extent. Specifically "Door" and "Wall" were eliminated as possible interpretations of the regions Chairback, Pic, and WBSK1, leaving those regions with unique interpretations.

The relational constraint, (NOT* (Adjacent Picture Door)), is self-explanatory. It introduces a clause reducing the likelihood that a region is a Picture by an amount proportional to the likelihood that adjacent regions are thought to be Doors. A loose definition of adjacency has been adopted in order to utilize this constraint in a partially segmented scene: two regions are adjacent if the line connecting their centers does not pass through a third region.

The numerical equilibrium likelihoods shown in Figure 52 were obtained by executing the evaluation functions as if they were Bayesian combinations of independent probabilities (i.e., a conjunction of likelihood values evaluates to the product of those likelihods, the negation of a likelihood to one minus the likelihood, and a disjunction of likelihoods to one minus a product of the negations of the likelihoods). More sophisticated quantification schemes are given in Ref. 58.

The final stage of analysis entailed searching for a set of unique interpretations that were globally consistent. The only ambiguity remaining in the global equilibrium context involved the regions Dr, LWall, and RWall, all of which still admitted both door and wall as possible interpretations. Homogeneity constraints forced LWall and RWall, both light colored regions, to take the same interpretation (either Wall or Door) and Dr, a dark colored region, to take the opposite interpretation. This basic ambiguity was resolved during the course of searching by the adjacency constraint on pictures, which led to a contradiction when either LWall or RWall was instantiated to Door.

The search proceded without any backup because the correct interpretation of each region already had higher likelihood than any alternative interpretations in the initial global context (Figure 52). The relative likelihoods of these correct interpretations increased monotonically with each successive correct instantiation. The final interpretations for the regions in Figure 49 are presented in Figure 53. A detailed trace of the reasoning, showing all instantiations and resulting reevaluations, appears in Ref. 58.

f. Conclusions

A working implementation of a new scene interpretation technique has been demonstrated. So far, the experimental results are inconclusive because of the simplicity of the test scene and the reliance on simulated range data for assigning

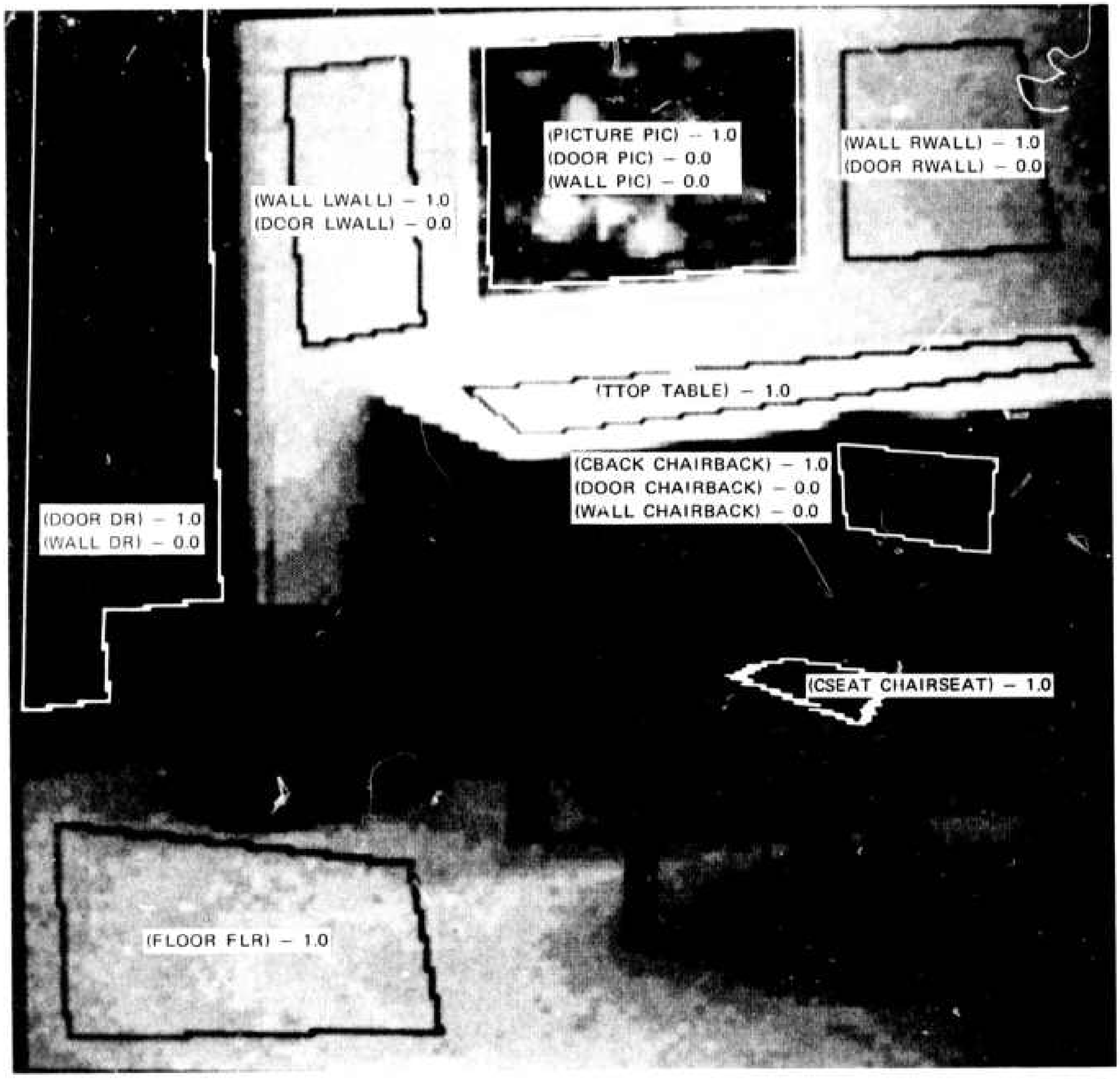


FIGURE 53 FINAL EQUILIBRIUM LIKELIHOODS FOLLOWING SEARCH

initial interpretations. The experiment will be repeated shortly using the actual laser range finder. Expectations are that the data will provide height estimates accurate to a few inches but that it will be too noisy to yield usable measures of local orientation. Consequently, additional constraints will be needed to distinguish horizontal and vertical surfaces at the same height, such as tabletop and wall. It would also be interesting to see how far one could get without direct range data by infering height and orientation entirely from pictorial clues such as image, height, and shadows.

Scene interpretation typifies a class of problem solving tasks involving large numbers of globally interacting constraints. A general constraint optimization algorithm is presented that, for sufficiently constrained problems, is more powerful than conventional AI search methods. This algorithm is implemented by representing interpretations and constraints as (simulated) asynchronous parallel processes. The resulting system organization features efficient data driven control and knowledge sharing.

PREVIOUS PAGE
IS BLANK

IV. SUPPORTING WORK

A. Introduction

A large amount of ancillary effort is required to support a project of this scope. Much of the programming in the project is done in our special AI language QLISP. (See Ref. 20 for a general discussion of some of the new AI languages.) Several additions to the power of QLISP as well as routine maintenance have been performed by B. M. Wilber and Daniel Sagalowicz. These are described in Section IV.B below.

In Section IV.C we discuss supporting hardware work. Bert Meyer has been responsible for the development of A/D and D/A interface equipment and programs connecting CBC gear to a PDP/11 computer and the PDP/11 in turn to our main PDP/10. A. E. Brain has continued his work on our scanning laser range finder.

Finally, in Section IV.D, we present a short summary of the current status of the SRI Artificial Intelligence Center computer system.

B. QLISP

The Artificial Intelligence Center uses QLISP [18, 20, 35, 67] in a variety of applications and makes it available to users outside the AIC for other purposes. QLISP extends INTERLISP [68] to provide powerful features for use in artificial intelligence applications. It provides a context mechanism for hypothetical reasoning and access to planning spaces; it provides automatic backtracking so that planning can pinch off fruitless considerations; it provides pattern-directed function invocation to simplify the programming of case breakdown; and it provides pattern-directed data storage and retrieval to permit symbolic access to the data base at a high level of abstraction.

Effort spent on QLISP has required both intensive work to improve QLISP's behavior within its scope and extensive effort toward extending it. The interval of this report includes three major accomplishments. QLISP and the INTERLISP compiler have been made to understand one another, permitting a substantial speedup of programs using the system. The pattern matcher has been replaced with a unification program providing enhanced power and flexibility at a reduction in storage space and computation time. We have designed and partially implemented a replacement for the data storage and retrieval mechanism, which we hope will provide both true associative access and allow graceful extension to much larger data bases than can be stored in any less drastic extension of the current data storage system. Finally, since QLISP is continually developing and responding to new needs, we have an ongoing program of fixing new and old bugs and providing miscellaneous other extensions. All four areas of concentration have contributed greatly to the utility of the system, while only the second two of these areas have noticeable extensive components.

A programming system can be more or less tractable depending on the cleanliness of the face presented the user. We have maintained a consistent effort to ensure that QLISP "looks reasonable" to both naive and sophisticated users. At one extreme, for example, QLISP is almost completely invisible to the user (or program) using none of its special features. Of greater interest, however, is the face QLISP presents to a user (or program) using its features. The extreme flexibility of INTERLISP has permitted us to build a highly automated interface between it and QLISP. A prime example of our efforts to maintain a clean user interface appears in Subsection IV.B.1 just below but a few points may be briefly made here. Even at the TENEX level, access to QLISP is no more difficult than access to LISP. QLISP will not intrude on a user who ignores it, but the user can freely intermix LISP and QLISP with no need to specially declare the presence or absence of QLISP in any segment of the program in question; QLISP system routines will be called automatically when appropriate and at almost no other time.

The rest of this section discusses the work we have put into QLISP since the beginning of 1974. Each major aspect of this work is treated in a separate subsection, although the reader should recognize that there is inevitably some degree of interaction and overlap between the contents of the various subsections. The "staff" of the QLISP effort has turned over completely during this interval, with Earl Sacerdoti and Rene Reboh turning their efforts elsewhere, while Michael Wilber and Daniel Sagalowicz now do most of the work. The current level of effort is almost two full-time people, but we expect this to diminish considerably as the current period of intensive development ends.

1. Compiling User Programs

One of the great shortcomings of QLISP was overcome toward the end of 1974: users could not compile a program containing any QLISP. Because of this conspicuous lack, QLISP users tended to use QLISP only for developing their algorithms and then, just for the sake of speed, recode their routines at no small expense of their time to use LISP where it could be made to do just as well. We have interfaced QLISP to the compiler in a manner about to be described. The resultant speedup of user programs has made QLISP tractable to many applications in which people formerly programmed around QLISP either by that kind of reprogramming or by avoiding its use altogether. The QLISP system is heavily enough used that we can presume the speedup to have enhanced the overall performance of the computer system with the result of making more computer time available to all, but the number of variables precludes our having any good statistics with which to back up this conjecture. However, this improvement alone has helped make QLISP the standard programming system for a large part of the CBC system.

QLISP was extended to compile user code by redefining two of the interior functions of the compiler. In one case, the modification was inessential in concept; we simply provided automatic tracing of compiled functions as a parallel to the automatic tracing provided for any QLAMBDA function defined by any other means.

The other, however, was essential for this extended notion of compilation to work at all; it entailed the translation of any QLISP constructs to LISP code that would call routines internal to the QLISP system. First of all, the structure of LISP's compiler required the translation of QLAMBDA functions to the corresponding LISP code as a special case. The body of the function is translated in almost the same manner as during evaluation of an interpreted function containing QLISP. When QLISP code is interpreted, we can (and in fact prefer to) translate only those parts of the function actually being used. This avoids unnecessary expenditures of time to translate possibly large amounts of code that may never get used before the translation must be discarded for any of a number of reasons. On the other hand, when a function is compiled, its entire body must be translated in order that the compiler generate the proper calls into QLISP. Finally, a small number of flags is used to determine whether to take the time to translate the function at all; some flags are set by the system, and some are provided for the user to indicate the presence or absence of QLISP in specific functions.

There was a bit of a challenge to be met in maintaining the compiler's user interface after adding the QLISP compatibility. In order that users need not become experts in the inner machinations of INTERLISP or of QLISP, we needed to let them compile, redefine, edit, and run individual functions or files of functions in their interpreted or compiled forms in any combination made available by INTERLISP without any need to make any special allowances for the possible or known presence or absence of QLISP anywhere in the code in question. On the other hand, this flexibility should not be provided at a cost in the overall system performance; speed bears just as heavily on the appeal of a system to users as do flexibility and unobtrusiveness.

Unfortunately, we could not simultaneously provide speed and unobtrusiveness in the case of the compiler interface. The compiler has provisions to automatically call a function supplied by the user (i.e., QLISP) for most of the places where QLISP is likely to occur, but unfortunately, that mechanism will not handle all the cases a user could generate. In this case, we were faced with no alternative but to intercept every function before it wa compiled and make sure all QLISP constructs were translated. This process approximately doubles the time taken to compile any given function. This is unavoidable if QLISP is actually present in the function being compiled, but it is entirely gratuitous if the function uses only LISP. We solve this problem (to the extent it is soluble) by providing the system of flags alluded to previously. The default state is that QLISP is suspected to be everywhere; we assume that the reason the user is using QLISP at all is that he has QLISP somewhere in his code, and as long as the user does not tell us where, we must look everywhere. The user can of course change the default state, but it is probably more apropos to indicate just where to look for QLISP; to this end, we provide more flags to indicate the presence or absence of QLISP in a whole file or in specific functions in a file, with each indication overiding the less precise ones.

Finally there was the problem of preserving the tremendous flexibility of INTERLISP, which permits the user to load (or refrain from loading) whatever combination of compiled and interpreted functions or files of functions that might be appropriate to production runs, debugging, compilation, development, or massively altering the QLISP program. This one was easy; it just required that we realize what needed to be done and exercise sufficient cleverness in doing it. As it turned out, the implementation was not difficult. All we needed to do was install a routine in the standard INTERLISP file package to put the symbolic file into the proper format before it was written out. This routine makes slight alterations in the format of the file from the format the user may have specified and, of course, it can cope with almost anything the user can specify, especially the undoing of its changes from a previous generation of the file.

The reader unfamiliar with INTERLISP should realize that we had to spend relatively little time in actually performing this surgery on the INTERLISP compiler or file-maintenance package. All the access points we needed were available to us. Some were available as features deliberately provided for applications such as ours. For example, the translation can be forced by setting a flag for the compiler, and it will automatically call DWIMUSERFN on encountering something it does not recognize; anybody can define new commands for the file-maintenance package simply by putting an appropriate entry on PRETTYDEFMACROS. Other access points are available simply because of the structure of INTERLISP; we get a first look at a function being compiled or at a symbolic file being written by the artifice of moving the definitions of certain critical functions to other places and installing in their stead our own functions, which do the preprocessing before calling the saved definitions.

2. Unifying Pattern Matcher

At the end of 1974, the pattern matcher in QLISP was replaced by a unification program, which increased the power available to us in the pattern matching operation. The unification principle is explained in [69] and [70]. When appropriate, we now unify two expressions rather than matching a pattern to an expression. The distinction is one of symmetry. With unification, any variable in either expression is free to take a value, and all variables with a given name must take the same value in both expressions for the match to succeed. (Formerly, values could be assigned only to variables occurring in the expression designated the "pattern".) For example, the pattern (CONNECTED PUMP ←X) will not match the expression (CONNECTED ←Y PLATFORM) because the variable X can take no value to make the pattern match the target expression. However, if the pattern is just taken as another expression to be unified with the target expression, the two expressions will unify, with $X and $Y taking the values PLATFORM and PUMP, respectively.

The unification program is far easier to maintain than the old pattern matcher, and preliminary measurements indicate that despite its greater power, it is a more compact and slightly faster program. While this change is too recent to allow us to have exploited it extensively, the next few paragraphs outline some of the uses to which we feel it could be put. Readers familiar with QLISP should note that in this discussion of the unifying pattern matcher, we will use the term "variable" in a restricted sense: since we are concerned with the assignment of values to variables to the exclusion of any concern with the instantiation of a variable to retrieve its value, we will use the term to mean only those variables that are expected to take values (and are written like ←X in QLISP), as opposed to those that are expected to already have values (and are written like $X).

The symmetry characteristic of pattern matching in the new regime can be used in several ways. Perhaps the most obvious is to construct expressions using variables supplied by the program in conjunction with items supplied to the program as data. For example, to find the difference between (CLASS A B C) and (CLASS A B D), one need only unify classes built by adding distinct variables to the two classes. Then each of the two variables will take for its value the extra item in the other set, as is necessary for the two constructed classes to unify. If the two classes above are supplied as the values of $I and $J, then (MATCH (CLASS $$I ←←X) (CLASS $$J ←←Y)) will store D and C as the values of $X and $Y, respectively.

Our new symmetry gives us a facility lacking in LISP (and heretofore also in QLISP) due to the traditional conception of LISP as a mechanization of a formal logical theory. Originally LISP had the notion, as did most formal mathematical theories, that the evaluation of a function generate precisely one result; if several answers are to be returned, they are all coded into the single result, sometimes giving schemes rivaling Goedel numbers in obscurity. With the emerging concept of LISP as a programming system, functions would sometimes pass their results back through free variables whose names were either given as part of the specification of the function or (awkwardly) passed as arguments. Now that QLISP variables in the expression supplied a QLAMBDA function can be assigned values as a result of the pattern match on entering the function, parts of the QLAMBDA pattern can be given as values to those variables during the match. A more interesting case arises when a variable in the supplied expression is matched to a variable in the QLAMBDA pattern. Then, when the pattern's variable is assigned a value in the body of the QLAMBDA function, the value is passed back to the supplied variable, giving QLISP a facility analogous to the "call by address" concept of the Algol thinkers.

The reader familiar with QLISP may have noticed a slight problem that arises, however, in precisely the case that an expression containing variables is supplied as data to a QLAMBDA function. The crux of the problem is that a QLAMBDA implicitly declares a new local context for any variables occurring in its pattern when the pattern is matched to the data expression supplied it. Since variables that might occur in the data are independent of variables in the program, even though they may have the same names, we should regard them as unrelated. Of course, the way we handle that case is to enter the (new) pattern matcher with an argument specifying that it is performing a QLAMBDA match. Then the unifier--i.e., the (new) pattern matcher--becomes asymmetric in the sense that any variables in the "data" expression are regarded as distinct from any variables occurring in the "pattern" expression that may have the same names and receive their values in the context outside the body of the QLAMBDA function. This procedure is known as "standardizing variables apart" in more formal discussions of the unification principle.

Unification can also be used to get the effect of storing universally quantified statements in the data base. If an expression with variables is asserted and then a query with variables is performed, and if the query can successfully match the asserted expression, then the result of the match may restrict the query in such a way that it gains information. For example, if one asserts that for every X, X+1 is greater than X and then later one has a symbol A and desires something greater than it, the proper query will produce A+1; in QLISP, it looks like this: after performing the assertion (ASSERT (GT (PLUS ←X 1) ←X)) and then later the query (IS (GT ←Y A) UNIFYING), then (PLUS A 1) will be stored as the value of $Y. Such a query presents no problem to the unification program, of course, but the access to the data base must be much more extensive to allow for such a case because everything in the data base must be retrieved if it has variables in it that might let it unify with the query. (The more usual data base access is far cheaper because the variables in the query must be matched to constants in a data base item for the item to be retrieved.) Thus, because of the relative expense and infrequency of this kind of query, we require the inclusion of the word UNIFYING in the query statement.

The user interface considerations for the unifying pattern matcher are somewhat different from those for the compiler. Speed, flexibility, and noninterference are the criteria here too, but at least we do not have to allow for reasonable behavior in the face of as wide a range of user behavior. Such a consideration has again shown up: the implied generality includes a very expensive case of relatively little interest, and so we require the user to specifically indicate the appropriateness of the "bad" case. Here, we prefer that the general case be the default; however, the marginal cost is so great that the marginal utility is very small. Therefore, we specify the defaults here to be just the opposite of what they are in the case of the compiler. There the cost is a mere factor of two in an operation (i.e., compiling) that is not done too often, and the benefit is that everything will be handled correctly. On the other hand, the general handling of this "bad" case would significantly increase the expense of retrievals from the data base, a frequent operation, and user programs would usually realize no benefit. Of course, another aspect of the face QLISP presents to its user is exemplified by the fact that these modes of operation are built into the system only as its default choices for the case that the user does not indicate a preferred mode of operation. This is why we can be so apparently cavalier about a factor of two; if the guess causing it is incorrect, the user can easily correct it.

3. Data Storage

As another part of the QLISP development, we are currently working on a new data base organization. In the next paragraphs we are going to present successively the old data base organization, called the Discrimination Net, and its possible deficiencies; the ideas behind the new organization and its possible advantages; and finally some of the deeper details of the new organization. This new data base organization is in a stage of development and about three man-months have been spent on it, consequently, the details are still sketchy and it is still too early to say whether the justifications presented in the next paragraphs are confirmed by experience or whether some major problems are going to appear that cannot be overcome.

a. The Discrimination Net--Organization and Discussion

The discrimination net organization is based on what is known in the literature as the "trie" technique [71-73]. Our own implementation was devised by Jeff Rulifson and greatly optimized by Rene Reboh. Although it is not the purpose of this report to present the details of the discrimination net, we think it would be useful to explain it in general terms with a few very trivial examples. Let us suppose that a user executes the following QLISP-like statement: (ASSERT(CONNECTED PUMP PLATFORM)in-context

C1). This inserts the record R1: {NETEXPRESSION(CONNECTED PUMP PLATFORM)[CONTEXT C1(MODELVALUE TRUE)]} into the data base. Since there is only one record in the data base, the organization tree (also known as the discrimination tree) has only one node that points to the (unique) record R1. Then the user executes the following QLISP-like statement: (ASSERT(DISCONNECTED PUMP PLATFORM)in-context C2). Then the data base gets a second record: R2: {NETEXPRESSION(DISCONNECTED PUMP PLATFORM)[CONTEXT C2(MODELVALUE TRUE)]} Now the discrimination tree has two terminal nodes , one pointing to R1 and one pointing to R2. But to enable the retrieval operations to distinguish easily between these two nodes a third one is created; it is only a discrimination node with just the information necessary to distinguish R1 and R2. In this example the needed information is that the two records differ by the first element, which in the first case is CONNECTED and in the second case it is DISCONNECTED--as indicated in Figure 54.

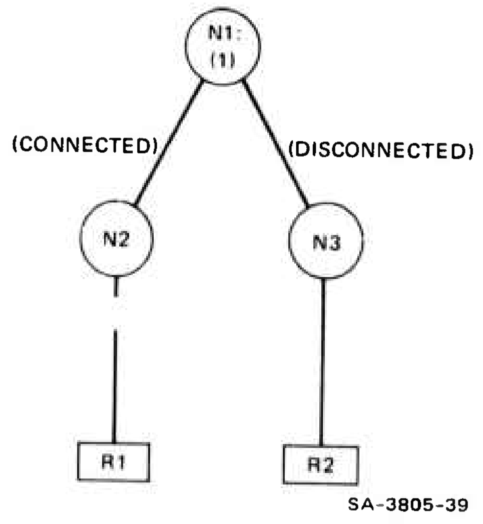


FIGURE 54 A DISCRIMINATION TREE WITH TWO NODES

If now a third assertion is executed such as (ASSERT(DISCONNECTED PUMP PUMP-PULLEY)in-context C1), then a third record is created, and the discrimination tree now has three terminal nodes and two discrimination nodes, as follows. The first node is as before and distinguishes between R1 and the rest. the second one distinguishes between R2 and R3. Figure 55 shows the result and is self-explanatory.

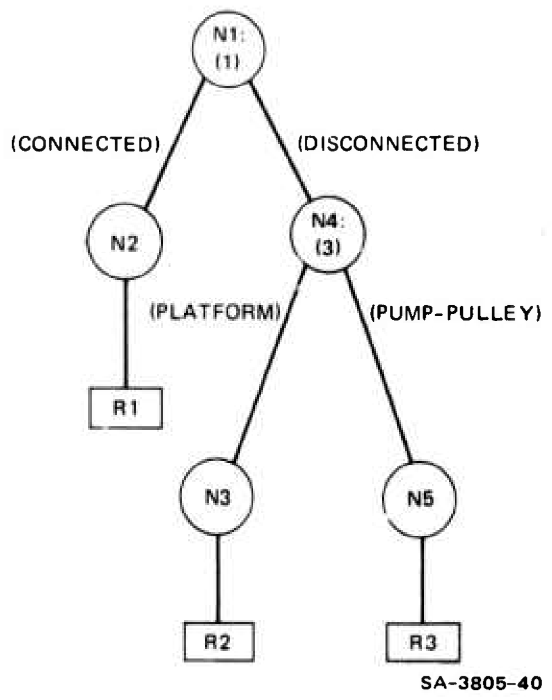


FIGURE 55 THE SAME DISCRIMINATION TREE WITH A THIRD NODE ADDED

Already at this point a number of remarks may be made without going any deeper into the details of the Discrimination Net implementation. First, each record is in fact composed of two parts: its "name" proper (e.g., (CONNECTED PUMP PLATFORM)) and a list of triples of the form (context-name property-name property-value) (e.g., (C1 MODELVALUE TRUE)). It is important to notice that the discrimination tree itself, which is used for data retrievals, does not include any information about the properties. This is due to the basic philosophy of the discrimination tree, and such information cannot be added to it in any way: once a record is attached to a terminal node it remains attached to that node no matter how its properties are modified. Said in other words, the discrimination is only based on syntactic information and not at all on semantic contents. The testing of the property values will still be done, but as an "after-pass"; i.e., after all the records that pass the discrimination testing have been assembled together. This may correspond to a waste of time, which could theoretically be avoided with a different scheme if that scheme were to use the semantic information to aid in that discrimination. In other words, we are looking for a scheme that keeps both semantic and syntactic information in the structure that is going to be used in the data retrievals.

A second important remark is that the semantic contents of the data base bears very little influence on the way the discrimination tree develops itself. It is even amusing (if not very profound) to notice that for the same records stored in the data base, the organization tree may be very different depending on the order in which these records were stored. One of the consequences of this fact is that the amount of work needed to do a retrieval is more or less independant of the semantic contents of the data base, but depends heavily on the number of nodes in the tree, or more precisely on the width and depth of the tree. To put it differently, after a long data base development, it can be expected that on the average, every retrieval will be long, since it will have to go through a great number of discrimination nodes, and this complexity will be more or less independent of the number of reasonable candidate records resulting from a given retrieval request. This last point is important, since it is generally felt that a good retrieval scheme should have an expected retrieval time roughly proportional to the size of the retrieval result.

The third criticism of the Discrimination Net is also related to the lack of semantic information in the organization of the discrimination tree. Because of this lack, there is no way for the language to decide how the data base should be organized on peripheral mass storage files, when we need to overlay the data out of the main memory. This task will somehow have to be left to the users. Concerning this last point two remarks should be made. First, in the environment in which QLISP has been known to operate in the past, i.e., the research environment, a strong case may be made that most users do not know how their data should be organized on peripheral files in order for their programs to be efficient. Moreover, even if each of them knew, the fact that the same data base is used by various users of different needs would necessitate that these users get together and decide collectively how to organise the data base; the result would be a division of the data into a great number of small "buckets" in which the records are strongly related in a semantic sense. It is our contention that such a subdivision is going to be difficult to create unless it is done automatically by the data base storage mechanism, in our case QLISP itself. The point is that the Discrimination Net does not give QLISP any help whatsoever in that direction.

It would be unfair not to mention at this point some of the possible advantages of the Discrimination Net. The main one is that it works and has been working satisfactorily, if not efficiently, for the past few years. The second one is that it probably requires less storage than anything else we could devise. And finally, all the disadvantages mentioned above are subject to strong discussion and will have to be measured in comparison with whatever other scheme we may implement.

b. Proposal for a New Data Base Organization

After an extensive search of the literature, it appeared to us that only two types of data base organization could be attractive for QLISP to use; one is the trie technique, of which the discrimination net is one example; the other is the inverted index technique. Essentially all the trie techniques would have the same types of disadvantages as the Discrimination Net, and many of them would not be attractive for QLISP. Thus it was logical to try the other possible type.

Let us quickly explain by an example how the inverted index works. The records stored in the data base are the same as in our example used for the Discrimination Net. The first record R1 is stored in the same way, but in the data base we add a new structure called the inverted index which has a number of nodes called CONNECTED, PUMP, PLATFORM, CONTEXT-C1, MODELVALUE, and TRUE. Under each of these nodes we put a pointer to R1. When the second record R2 is added we create two new nodes in the inverted index called DISCONNECTED and CONTEXT-C2. Moreover, we put a pointer to R2 under the nodes DISCONNECTED, PUMP, PLATFORM, CONTEXT-C1, MODELVALUE, and TRUE. Now some of the nodes have two pointers , some have only one. After the third record is added, the inverted index looks like this:

```
CONNECTED: R1
DISCONNECTED: R2 R3
PUMP: R1 R2 R3
PLATFORM:  R1 R2
PUMP-PULLEY: R3
CONTEXT-C1: R1 R3
CONTEXT-C2: R2
MODELVALUE:  R1 R2 R3
TRUE:  R1 R2 R3
```

We keep building the data base in this way. When a retrieval is attempted the algorithm is quite simple, at least in theory: if we are looking for every record that mentions the PLATFORM, we look in the inverted index for the node PLATFORM and find these records immediately. If we look for all objects connected to the PLATFORM in context C2, we take the intersection of the nodes: PLATFORM, MODELVALUE, TRUE, and CONNECTED. All this is very simple and quite well known, but it has one major inconvenience. Taking the intersection of big sets is a long operation, and the length of that operation is not related to the size of the result but to the size of the sets with which one started. This is considered to be a major disadvantage of such a technique. In our example, to find all the objects that are connected to the PUMP-PULLEY in context C1, we would take the intersection between five nodes (CONNECTED, PUMP-PULLEY, CONTEXT-C1, MODELVALUE, and TRUE) to find out that there is no such object; this is a very troublesome loss of time.

To alleviate this problem, we can use the technique of subdividing the indices by forming some of the interactions when data are stored. Then, the index at any given node may itself be indexed by some other index nodes before yielding a list of records. In our example, the index for DISCONNECTED could be subdivided, with the next level of indexing distinguishing CONTEXT-C1 from CONTEXT-C2. The result would be that under each index the list of records would be shorter and the intersections easier to do. The indices that have been subdivided would point to their subdivisions. The question then is: which indices should be subdivided? If they are subdivided to the point where they cannot be subdivided anymore, then we would create a huge number of indices and the inverted index structure would be bigger than the rest of the data base, which clearly would be inefficient. On the other hand, if we subdivide them only to a predefined depth, then there will always be some indices on which the intersections would be too costly.

In the implementation we are thinking about, we have solved this problem in the following way. Under each index we keep account of the number of records it points to; when this number becomes too large--i.e., bigger than a given threshold (which is user modifiable)--then the index is subdivided. In this way we guarantee that no intersection will occur over sets that have a size bigger than the threshold. Of course we also give the user the possibility to declare that some indices must always be subdivided or must never be subdivided.

The influence of the threshold on the way the data base operations behave is not a simple one. When the threshold is small, the number of indices becomes large and therefore the inverted index list becomes large; also the time taken by the intersections becomes small, but it becomes more difficult to get to the point where the actual intersection occurs. On the other hand, the data base operations that end up taking unions of sets of records definitely become longer. It therefore will be necessary to measure whether the advantage of faster intersections outweighs the other disadvantages. It may be that for a given program there is an optimal threshold, and the difficulty will be to find what it is.

In this new data base organization the user has the choice of the "subdivision threshold" as we have just seen above. However, this is not the only freedom he has. At the beginning of this subsection, we saw that the "basic indices" are the elements of the record name, the context, the property names, and values. In fact, this does not need to be so. The user will be given the choice of deciding what are his basic indices. In some cases, he may decide that a property name or value is not needed as a basic index, mainly because no associative retrieval will ever be done on it, or that it will be so rare that the disadvantages of having it as a basic index outweighs its advantages.

At this point we have given the general ideas of our new implementation and will leave some of the more intricate details for the next subsection. It is now time to explain why we feel that we may have solved some or all of the problems we think occur with the Discrimination Net. The main point is that now the "inverted list" includes both syntactic and semantic information. All of these may be used during associative retrieval and no additional pass is necessary for the semantic information testing alone.

The second point is that now the retrieval time is roughly proportional to the size of the result of the associative retrieval which, as mentioned earlier, is a sought-after characteristic of "good" associative retrieval techniques. With respect to this point, it should be noted, though, that we do not know at this time whether the associative retrieval time is not always too long, and in particular always longer than the corresponding time used in the Discrimimation Net technique; if this is the case, obviously the whole thing would be a failure.

The final point concerns the problem of storage on peripheral devices. When a new record is put into the data base, the question will be where to put it outside of main memory. As mentioned previously, a reasonable answer is to put it in a "small bucket" with other records that are strongly semantically related. Such a bucket may just be the list of records that are pointed to by the smallest subindex list on which the record is also located. The fact that this list is the smallest may very well indicate that this subindex has a strong semantic meaning, and this may be enough to have a reasonable peripneral file organization. This point will be investigated further if the new data base organization appears to be competitive with the Discrimination Net.

In summary, it appears to us at the present time that this new organization does satisfy the objectives that we specified in the first subsection; however, there are still two big unknowns that will be resolved only when the implementation is completed to a fairly advanced point. First, it is not clear what time improvement we are going to get; second, it is not clear how much more space will be required.

c. Some Additional Details on the Inverted Index

In subsection b above we gave the general ideas on the new organization; however a lot of details interfere and make the actual implementation much more intricate. As the reader may know, QLISP has three different types of data: tuples, bags, and classes. (We do not mention vectors, which for the present discussion are the same type as tuples). We will quickly define these three data types by their characteristics: tuples are the n-tuples of ordinary mathematics; bags are its multisets (loosely, sets that can have repeated elements); and classes are its classes (or sets). For example: (TUPLE A B) is not equal to (TUPLE B A), (BAG A B A) is equal to (BAG A A B) but not to (BAG A B), (CLASS A B A) is equal to (CLASS A B). These various equalities or inequalities have to appear correctly in the way the indices are chosen. If (TUPLE A B) is put into the data base we need more than just the three indices TUPLE, A, and B because this would not distinguish it from (TUPLE B A). Similar remarks apply to bags. The solution adopted is as follows (in spirit but not quite in realization).

(TUPLE A B) corresponds to the indices:
1TUPLEA 2TUPLEB 3TUPLE (this last one corresponds to the length of the record).

331

(BAG A B) corresponds to the indices:

1BAGA 1BAGB 3BAG (the length again).

(BAG A A B) corresponds to the indices: 1BAGA 2BAGA 1BAGB 4BAG.

(CLASS A B) corresponds to the indices:

CLASSA CLASSB 3CLASS.

Let us see now how a data retrieval is performed by looking at some examples. Let us consider a request of the type (TUPLE A ←X C) where ←X stands for "any element". This just means: find all records that have the indices: 1TUPLEA 3TUPLEC 4TUPLE, and is easy to satisfy. In the same way, the request (BAG A ←X C) corresponds to all records that have indices 1BAGA 1BAGC 4BAG. Note that in this last case some of the records may also have 2BAGA or 2BAGC as a possible index. Until now everything behaves nicely. Things become slightly more cumbersome with the next example: assume the request (TUPLE A ←←X C) where ←←X stands for "any number of arguments." (In other words this request is matched by (TUPLE A C) as well as by (TUPLE A B C D E F G C).) Then the matching records certainly have the index 1TUPLEA but their next two indices will be one of the pairs (2TUPLEC 3TUPLE), (3TUPLEC 4TUPLE), and so on. The associative retrieval becomes much more complex. (In the first implementation, we will just use 1TUPLEA as the index and forget about the fact that there is also a C at the end of the record). The similar case behaves very nicely with bags and classes and does not need to be mentioned.

In QLISP the "name" of a record, i.e. its syntactic content, may be any LISP expression; for example, (TUPLE A (TUPLE B C)D) is a perfectly legal QLISP expression to be stored in the data base. This is in fact stored as two records: (TUPLE B C) which is, say, record R1; and (TUPLE A :R1 D) where :R1 stands for "pointer to R1". As far as storing these records is concerned, the implementation is straightforward; the record (TUPLE A :R1 D) corresponds to the indices: 1TUPLEA 2TUPLE:R1 3UPLED 4TUPLE. The difficulties start with the retrieval. How do we satisfy the request: (TUPLE A(TUPLE B ←X)D)? There are two ways to satisfy it. We can start by finding all records that match (TUPLE B ←X), call them RX1 RX2 RX3 ..., then find which of the records (TUPLE A :RX1 D) (TUPLE A :RX2 D) ... exist in the data base. Or, we may try to directly find all records that match (TUPLE A ←Y D) and intersect the records that replace ←Y with the list (RX1 RX2 ...). In our preliminary implementation, we use the second scheme: we find all records matching (TUPLE A ←Y D) and, if there are not too many of them, we give all of them to the "pattern matcher" which will eliminate the nonmatching ones. If there are too many, we start all over and try the first scheme. The reason why we chose this implementation is that it was much easier to code and debug and we wanted something running fast to test the ideas; however, it is clear that this implementation is costly both in time and in space (by the unnecessary "conses" it generates) and if the new data base ideas look promising, this will have to be improved upon.

Finally, the last level of difficulty is caused by a totally different type of retrieval which is possible in QLISP. This can be called the pattern retrieval: a user may ask to find all patterns that match a given datum. For example, one might ask to find all patterns matching (TUPLE A B) and would expect to be given back records like (TUPLE A B), (TUPLE A ←X), (TUPLE ←Y B), and (TUPLE ←X ←←Y). We will refer to this as the pattern retrieval; this turns out to be the most difficult data base operation to implement. Let us try to explain how we attempted to solve this problem. In the example above, we are given the data (TUPLE A B); what are the possible indices of the matching patterns? A matching pattern could have 1TUPLEA but may not; it could have 2TUPLEB but it may not; it could have 3TUPLE but again it may not. In fact, at this point it looks as though we do not know anything about the patterns. That is not quite true, luckily enough; if the pattern does not have 1TUPLEA as an index it must have something like 1TUPLE←X or 1TUPLE←←X. If it has 1TUPLEA or 1TUPLE←X and does not have 2TUPLEB, it must have 2TUPLE←X or 2TUPLE←←X. However, if it has 1TUPLE←←X, then it may very well have nothing else, or rather it may very well have only 2TUPLE or 3TUPLE and so on. The algorithm to solve this simple pattern retrieval is already quite cumbersome; it becomes even more cumbersome when treating the following one.

Let us assume that we are given the following data and we are looking for the matching patterns: (TUPLE A(TUPLE B C)D). We then have the same problems as with the last example, plus a new one. We have to find all the patterns that match (TUPLE B C) and add that level of complexity to the algorithm. Here again two strategies may be taken. We may first do the pattern retrieval for (TUPLE B C) and then do the pattern retrievals for the things like (TUPLE A :RX D); this is conceptually simple but generates a lot of useless records RX and, therefore, a lot of useless pattern retrieval attempts. Or we may start by doing a pattern retrieval corresponding to (TUPLE A :R← D) where :R← corresponds to "any record with a ←X or ←←X in it" and then intersect the resulting list of :R← with the records matching (TUPLE B C). The latter alternative is essentially the attitude we are taking (with the same restricted implementation mentioned in the data retrieval part above). Adoption of this strategy causes storage of a new record, such as (TUPLE A (TUPLE B ←X)), to generate the indices 1TUPLEA 2TUPLE:R← 2TUPLE:R1 3TUPLE. The record R1, which is the name of (TUPLE B ←X) has the indices: 1TUPLEB 2TUPLE←X 3TUPLE.

There are, of course, a lot more details to the implementation than those just given. However, these details already give the reader a general feeling of the difficulties involved which did not appear in the general ideas mentioned in subsection b. It is important to know that these difficulties are present because they are the main cause of time and space inefficiencies. It is the solution of these details that will allow or disallow the whole idea to be a viable one.

4. Maintenance

Maintenance produces no milestones but is crucial to the utility of a developing system. We continually try to keep QLISP as current as possible; this includes fixing bugs reasonably quickly after they are found and trying to minimize the lag between the arrival of a new version of LISP and the appearance of the matching QLISP. While this task is nominal in extent, its requirements are often unpredictable and, since it can occasionally take up to half of one person's time (averaged over a month), it can sometimes have a noticeable effect on the progress of developmental tasks. However, the importance of this task leads us to give it a high priority most of the time.

We provide a fast turnaround on repairing bugs in the system and providing minor extensions. Users need wait only a couple of hours (and never more than 24) to continue using QLISP without needing to program around the flaws they discover. A high priority has also been given the release of a new QLISP to go along with every new version of LISP, so that QLISP users may begin using the new LISP as soon as possible after its arrival. Under normal circumstances, the job of loading a new QLISP and performing various associated ancillary tasks would take an hour or less, but in our case this time is sometimes increased to as much as several days by two special circumstances. First of all, the parts of QLISP directly concerned with its interface to LISP are much more sensitive to details of the inner structure of INTERLISP than most other LISP programs. For example, the functions in QLISP that replace parts of the compiler must have the same number of arguments (of the same names) as the standard functions they replace. Since INTERLISP is also in a continual state of development, such details will change from time to time. Also, new versions of LISP frequently have new features adapted to (and often motivated by) the constantly increasing requirements of QLISP; we try to use the new features right away, but if testing reveals they do not work, we must patch around them until the arrival of the next version of LISP. The processes of removing old patches, testing the resultant QLISP, and possibly installing new patches can considerably extend the time required to bring up a new QLISP.

We put a high value on upward compatibility; we always endeavor to restrict our changes to the kinds that extend the language without invalidating programs a user may already have. Success at this endeavor is required for us to retain any given user for any considerable length of time. Although this does not compete for much of our time with the other aspects of the QLISP effort, it is something we must always keep in mind as we change the QLISP system. Unfortunately, we cannot always maintain the continuity we wish. Sometimes overriding considerations dictate a change in the user interface requiring a change in QLISP programs. A program can produce correct results by virtue of a bug which masks a bug in QLISP and is masked by it. When the QLISP bug is fixed, the program bug takes its toll. We try to minimize those changes that are not upwardly compatible, but they unfortunately cannot be eliminated.

5. Summary

We have now described how we have directed the part of this project devoted to QLISP. Major results are in the areas of compilation of user programs, a unification pattern matcher, and a new organization of the data storage mechanism, while we also have continued an ongoing task of routine maintenance. In our undertakings, we have always kept sight of a higher goal: that of providing a reliable, dependable programming system that permits users to concentrate on their own programs with minimal distraction from bugs or other awkward characteristics QLISP may have.

C. Hardware Work

1. Interface Equipment

The interface configuration of the present CBC system is shown in Figure 56. The general approach is to use the local IMP and PDP-11 simply as an I/O processor for the PDP-10. All calculation and control will reside in the PDP-10, while the PDP-11 takes care of the localized real-time commands and control of the peripherals. All PDP-11 interfaces are implemented with standard DEC modules and hardware. The Analog-to-Digital converter is a

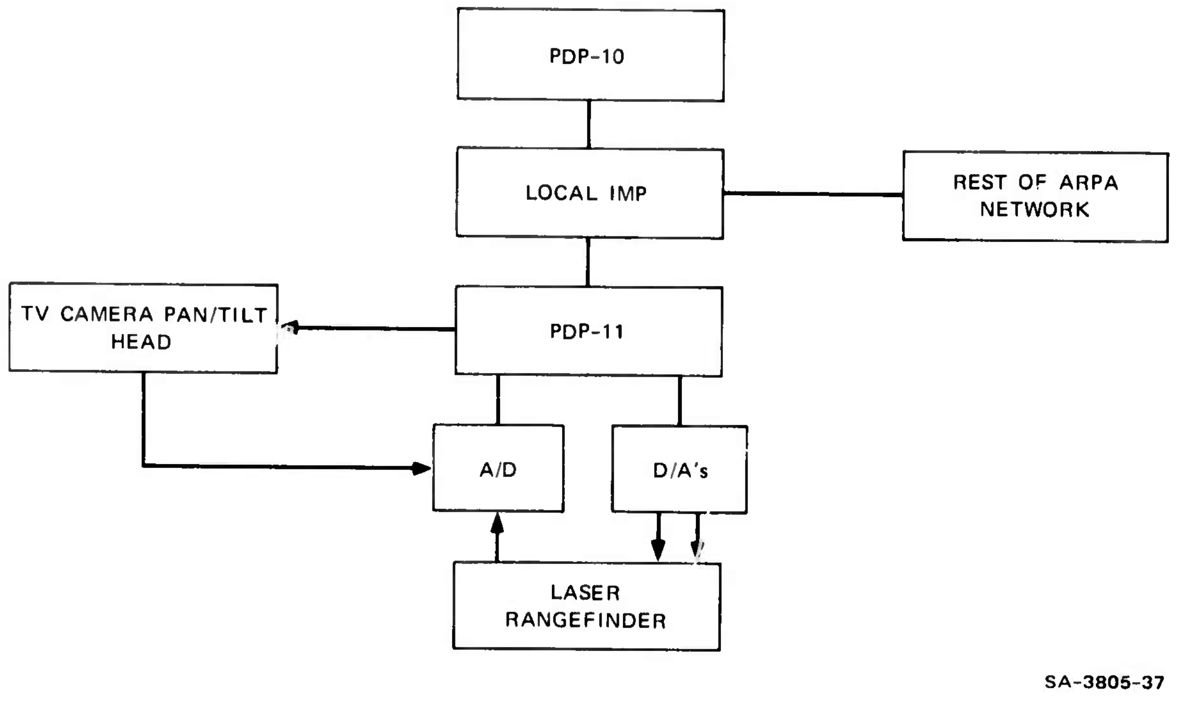


FIGURE 56 INTERFACE CONFIGURATION

multiplexed 12-bit device fabricated of standard modules. The Digital-to-Analog converters (also 12-bit) provide analog x-y position commands to the range finder.

In use, then, the PDP-11 does the following. When a LISP program in the PDP-10 wants to move the camera, it sends to the PDP-11 the desired pan and tilt angles; the PDP-11 then starts the pan/tilt head moving in the proper direction and, via the A/D converter, monitors its position. When either angle reaches the desired position the rotation is stopped, and when both angles are stopped a signal is sent to the PDP-10 to so signify.

If a LISP program desires a range measurement, it sends the desired angular positions to the PDP-11. The PDP-11 passes these values to the D/A converters. Since the laser deflection equipment has no position feedback, the PDP-11 merely waits an appropriate length of time and then starts the A/D reading the analog value (proportional to the range at that point) from the range finder. That value is sent back to the PDP-10.

Provision has been made but not yet implemented to add control of the TV camera's iris, focus, zoom, and color wheels, and also to add communication with the VIP-100 word recognition machine.

2. Laser Range Finder: Random Variation of the Range Measurements Arising From Imperfections of the Phase Measuring Equipment

a. Background

In a previous report [37] a discussion was given of random variations of range measurement arising from the finite number of quanta received by the photomultiplier tube during the time interval of a measurement. This is certainly the primary cause of range "noise," but it is not the only source: some of the observed variability can be attributed to the phase measuring equipment, and it is the purpose of the following investigation to provide data from which one may estimate that component of the observed variability. This is not to imply that the phase measuring equipment is defective either as functioning hardware or in design, but rather to recognize that the dynamic range over which measurements are being taken is well outside the design for the precision we require. In scanning a field of view that includes an object such as an air compressor, the input signal to the phase meter may cover a dynamic range of 105 dB, made up as follows:

Variation due to inverse square law	16 : 1	24 dB
Variation in reflection coefficient	30 : 1	29 dB
Specular reflection	20 : 1	26 dB
Surface orientation	20 : 1	26 dB

where the maximum signal comes from a specular reflection from a surface close in, and the weakest signal from a low reflectivity surface that is almost tangential to the scanning beam and at maximum range.

Figure 57, redrawn from the HP data sheet, shows the magnitude of the offset error to be expected in the output of the phase meter in relation to the dynamic range of the signal

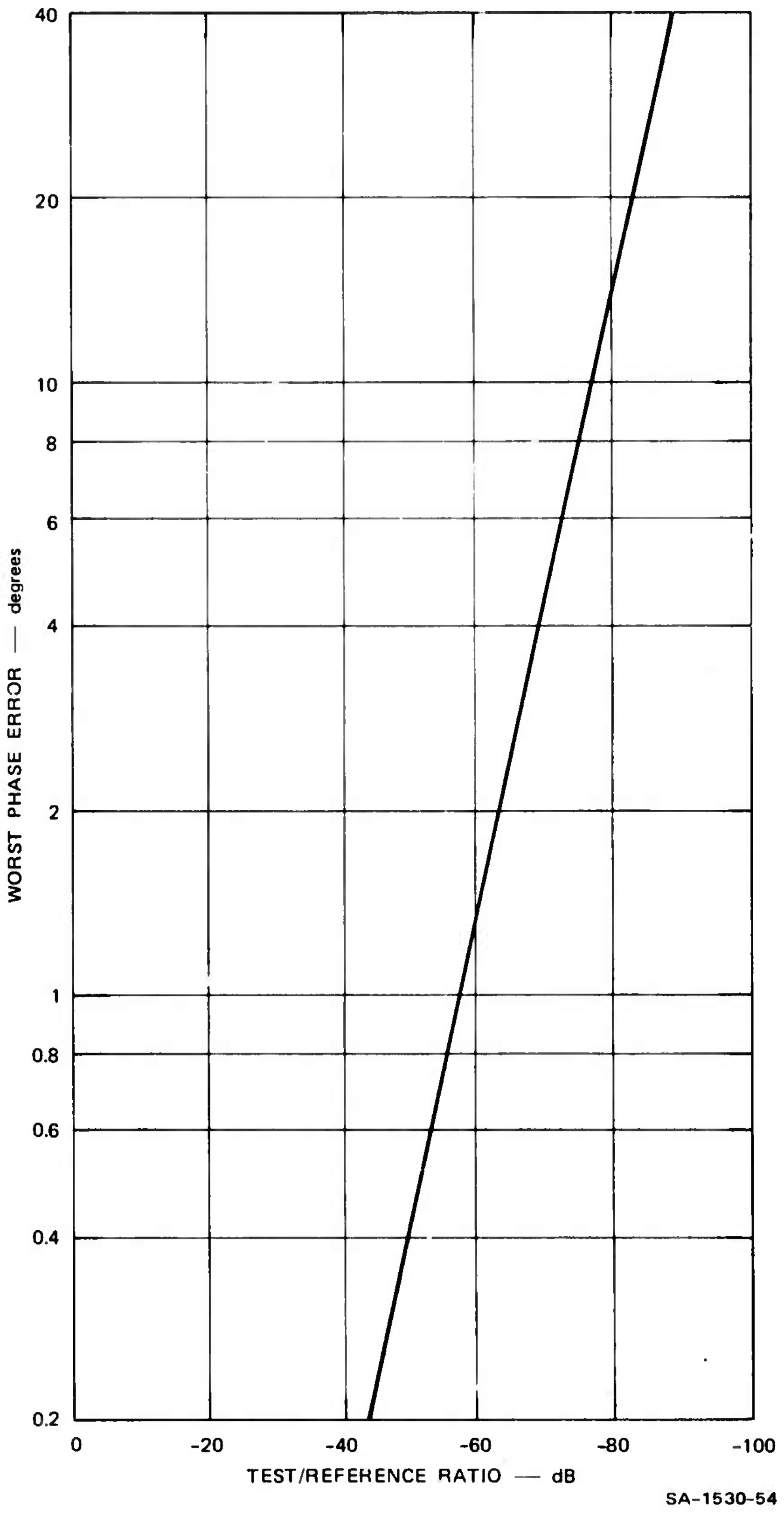


FIGURE 57 PHASE ERROR CHARACTERISTIC OF HEWLETT-PACKARD 8407A

being measured. This is a displacement error, not a random variation error ("noise" on the signal), and by implication is independent of the signal level, which from the nature of the device is rather unlikely.

b. Experimental Observations

Tests for phase constancy were made using the following pieces of equipment made by Hewlett Packard, and believed to be in normal operating condition:

8601A	Signal Generator	# 959 - 01335
8407A	Network Analyzer	# 959 - 00227
8412A	Phase-Magnitude Display	# 968 - 00276

As an initial check the laser ranger was used to take 300 range measurements of a point on a table top about 8 feet away--this represents an average level signal. The mean range was 96.88 inches, and the standard deviation was 0.607 inches. The laser had not been tuned for maximum power, and there was nothing unusual about the situation in general, so that it seems fair to describe the equipment as being in normal condition.

At this point, the two input leads coming from the photomultiplier and reference beam photodiode were removed from the test and reference inputs to the phase meter, and all of the subsequent measurements to be discussed were made using signals from the 8601A Signal Generator, the single output from which was converted to dual output by means of the power splitter 11651 - 60001. In operating the laser ranger it was observed that the two vertical slide attenuators on the Phase Meter, marked Display Reference dB, and which presumably affect only the display circuitry, nevertheless do interact with the measuring equipment and in particular affect the phase displacement error associated with limiting shown in Figure 57. (When scanning over a gray-scale step chart at constant distance, the range reading showed minimum offset error when the attenuators were at +10 dB and +2dB.)

Test No. 1 Identical Signals to Both Channels, as shown in Figure 58.

Input frequency 9.000 MHz Bandwidth 100 Hz

100 measurements at 30 millisecond intervals

Signal Level	Range (inches)	Standard Deviation
-10 dB	115.04	0.096
-20 dB	114.56	0.078
-30 dB	114.53	0.056
-40 dB	114.96	0.126

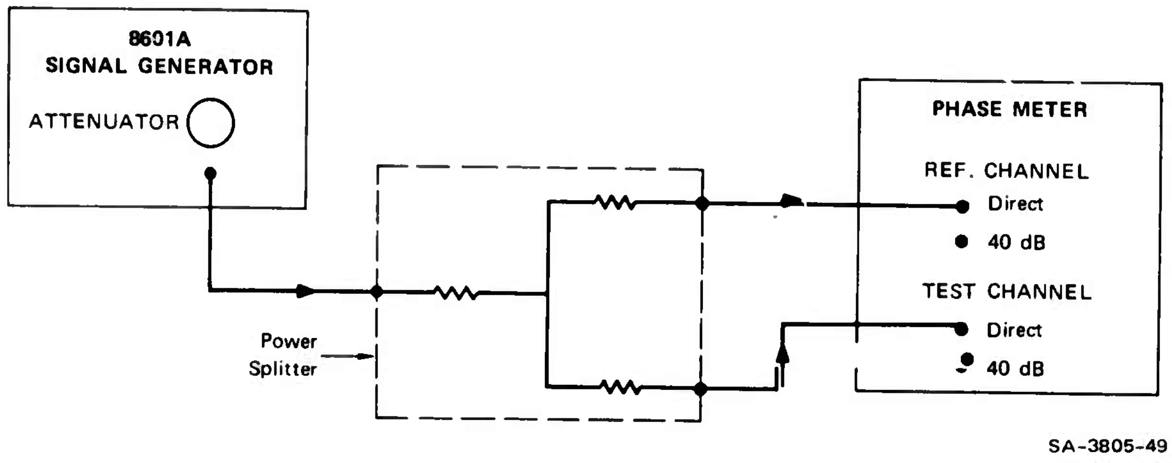


FIGURE 58 TEST 1: IDENTICAL SIGNALS IN BOTH CHANNELS

The -40 dB level corresponds to an input signal of 3 millivolts. One inch in range corresponds to approximately one degree in phase shift. Thus, for relatively high input levels, identical in both channels, the measurements are essentially noise free; however, the displacement of the zero by 0.5 inch for only 10 dB change in level is significant, and in the light of previous experience can probably be attributed to effects in the reference channel. It is desirable to adjust the reference level to read close to mid-scale on meter, and not merely keep it within the permissible range marked on the dial.

Test No. 2 Reference Channel 40 dB Above Test Channel, as shown in Figure 59.

100 Measurements at 30 millisecond intervals.

The reference level was maintained close to mid-scale by adjustment of the three position attenuator on the Phase Meter panel. This naturally caused displacement of the zero because of small differences of wiring lengths within the attenuator, but our primary interest was in measuring the "noise" level; i.e., the standard deviation of the measurements.

Test Level	Equivalent Value	Range (inches)	Standard Deviation
-20 dB	30 mv	114.70	0.099
-30 dB	10 mv	114.70	0.097
-50 dB	1 mv	118.28	0.103
-60 dB	300 uv	122.56	0.361
-70 dB	100 uv	124.80	0.758
-80 dB	30 uv	129.19	4.352

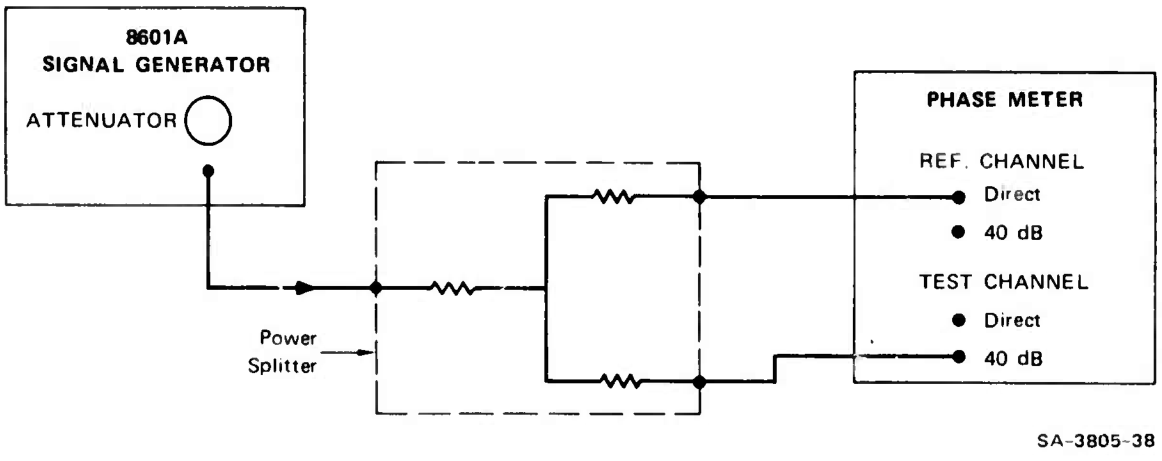


FIGURE 59 TEST 2: NORMAL CONDITION OF USE

From the above measurements it is evident that random variations due to the measuring equipment itself are significant for test levels below 300 microvolts. It is difficult to arrange for any test scene that includes the compressor to cover a range of less than 70-80 dB. Moreover, if the photomultiplier gain is increased to raise the low-level signals until they cause the phase meter to operate in the low-noise range, then the preamplifier will go into nonlinear operation on the specular reflections and will produce substantial positional errors. It is perhaps worth pointing out that the time delay of the signal created by a photomultiplier is not insignificant in the present context, and that this time delay is a function of the power supply voltage, which is used as the gain control. Fluctuations of the power supply voltage therefore modulate the phase difference between the test and reference channels, and are thus a further source of "noise" on the range measurement.

D. Artificial Intelligence Center Computer Facility

1. Major Accomplishments

Figure 60 shows the current PDP-10 TENEX computer hardware block diagram. The two major changes during this contract year were the total replacement of all existing main memory (which was very unreliable) with a new 256,000-word memory and the replacement of a five-drive disk system (which was also very unreliable) with a used nine-drive disk system that is considerably better.

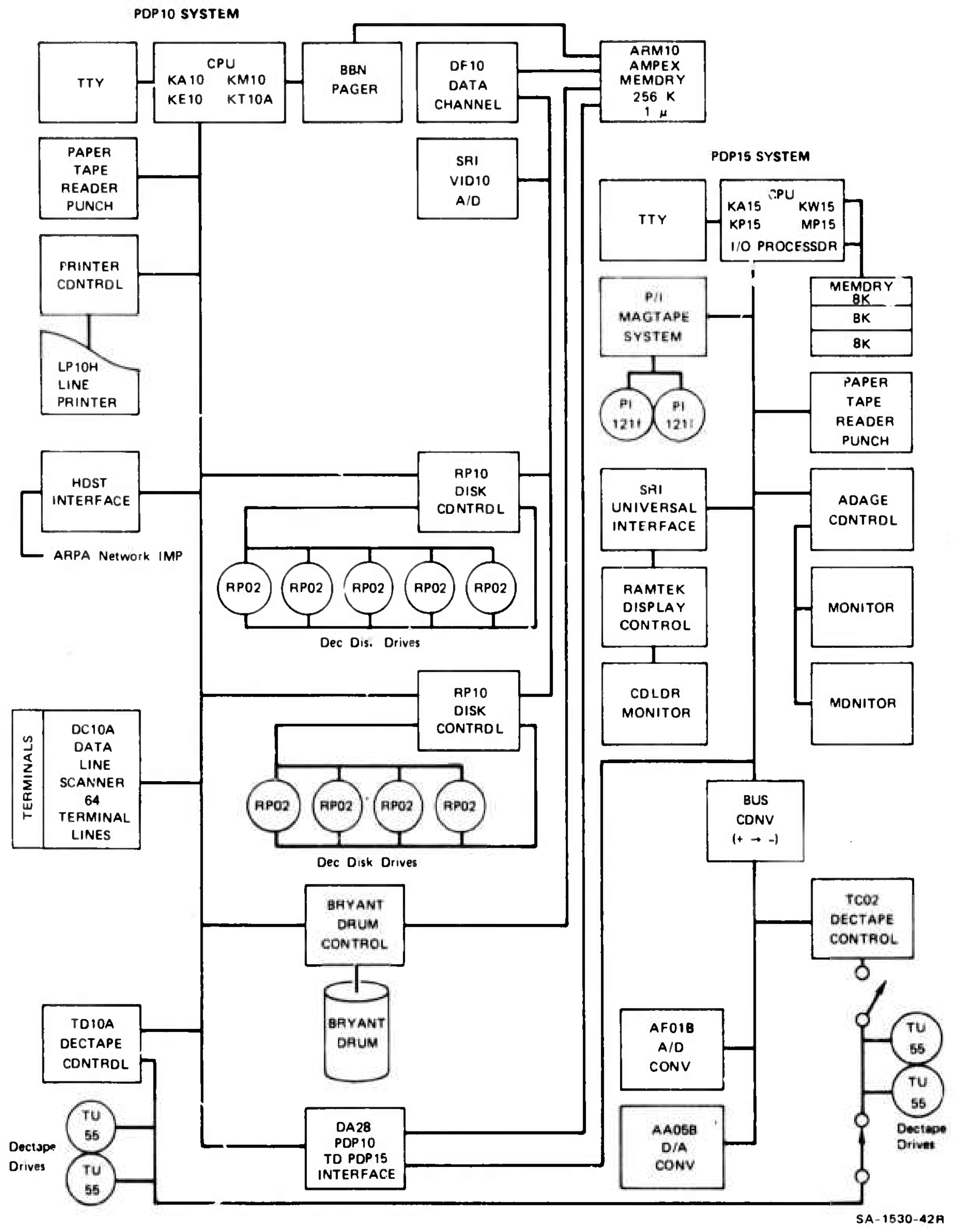


FIGURE 60 PDP-10 AIC TENEX COMPUTER SYSTEM

The operating system software has undergone a major restructuring in order to provide more CPU cycles to the users at a small loss in interactive response. The current hardware/software configuration now routinely delivers 85 to 90 percent of all CPU cycles to the user during periods of normal load.

2. Portents for the Future

The current computer facility is heavily loaded for about 80 hours per week. Most of our research programs are limited by our computer power and address space. Both of these areas are being investigated thoroughly in order to provide the computing service that the research staff needs.

PREVIOUS PAGE IS BLANK

V PUBLICATIONS AND PRESENTATIONS

A. Publications

Deutsch, Barbara, "The Structure of Task-Oriented Dialogs," Proceedings IEEE Symposium on Speech Recognition, Artificial Intelligence Center Technical Note 90, Carnegie-Mellon University, Pittsburgh, Pennsylvania (April 15-19, 1974).
Intelligence Center Technical Note 90.

Fikes, Richard E., "Deductive Retrieval Mechanism for State Description Models" (submitted to IJCAI-75).

Garvey, Thomas D., and Jay M. Tenenbaum, "On the Automatic Generation of Strategies for Locating Objects in Office Scenes," Proceedings 2nd International Joint Conference on Pattern Recognition, Copenhagen, Denmark (August 1974).

Hart, Peter E., "Progress on a Computer-Based Consultant Project," Artificial Intelligence Center Technical Note 99, January 1975 (submitted to IJCAI-75).

Hendrix, Gary, "Expanding the Utility of Semantic Networks Through Partitioning" (submitted to IJCAI-75).

Nilsson, Nils J., "Artificial Intelligence," Proceedings IFIP, Stockholm, Sweden (August 1974).

Sacerdoti, Earl D., "Planning in a Hierarchy of Abstraction Spaces," AI Journal, Vol. 5, No. 2 (Summer 1974).

Sacerdoti, Earl D., "The Nonlinear Nature of Plans," Artificial Intelligence Center Technical Note 101, January 1975 (submitted to IJCAI-75).

Tenenbaum, Jay M., "ISIS: An Interactive System for Scene Analysis Research," Proceedings 2nd International Joint Conference on Pattern Recognition, Copenhagen, Denmark (August 1974).

B. Presentations

April 1974 Stanford University Seminar

Jay M. Tenenbaum: "Reasoning About Scenes"

May 1974 Stanford University

Jay M. Tenenbaum: "Vision by Distinguishing Features"

May 1974 IEEE Systems, Man, and Cybernetics Society, San Francisco Chapter

Jay M. Tenenbaum: "Vision by Distinguishing Features"

June 10-12, 1974   Interactive Knowledge Systems Workshop, Alpine Meadows, California

Barbara Deutsch: "The Structure of Task-Oriented Dialogs"

Richard E. Fikes: "A Modeling and Planning System for the Computer-Based Consultant"

Gary Hendrix: "Modeling Simultaneous Actions on Continuous Processes"

Earl D. Sacerdoti: "The Procedural Net: A Representation for Reasoning About Actions"

June 1974   General Motors Research Laboratory, Warren, Michigan

Jay M. Tenenbaum: "Scene Analysis Research at SRI"

June 1974   Psychology Department, Stanford University

Jay M. Tenenbaum: "Perception as Problem Solving"

July 4, 1974   Department of Computational Logic, University of Edinburgh

Earl D. Sacerdoti: "Procedural Nets"

July 10, 1974   AISB Summer Conference, Free Session

Earl D. Sacerdoti: "The Procedural Net: A Declarative Procedural Representation of Knowledge"

July 15-16, 1974: Department of Computational Logic, University of Edinburgh

Earl D. Sacerdoti: "QLISP: Language for the Development of Complex Systems"

July 18, 1974   Department of Bionics, University of Edinburgh

Earl D. Sacerdoti: "Procedural Nets"

July 24, 1974   Department of Computational Logic, University of Edinburgh

Earl D. Sacerdoti: "The SRI Computer-Based Consultant Project"

August 9, 1974   IFIP Congress

Earl D. Sacerdoti: (Panelist at Session on Robotics)

October 14, 1974   University of Rochester

Richard E. Fikes: "Modeling and Planning Mechanisms for a Computerized Consultant"

October 16, 1974   SUNY at Buffalo--Lecture given at State University of New York

Richard E. Fikes: "Modeling and Planning Mechanisms for a Computerized Consultant"

October 17, 1974   SRI-Washington Office Seminar for Representatives from Government Agencies

Peter E. Hart and Nils J. Nilsson: "Computer-Based Consultants"

November 8, 1974   Stanford Artificial Intelligence Laboratory Representation Seminar

Earl D. Sacerdoti: "The Procedural Net: A Representation for Reasoning About Actions"

November 14, 1974   Computer Science Colloquium, University of Washington

Richard O. Duda: "Some Approaches to Computer Diagnosis"

November 21, 1974   The Eighth IBM Computer Science Symposium, IBM Amagi Homestead, Japan

Richard E. Fikes: "Data Base Requirements for Answering Questions in a Computer-Based Consultant"

December 6, 1974   Stanford Artificial Intelligence Laboratory

Gary Hendrix: "Encoding Knowledge in Homogeneous Semantic Networks"

January 22, 1975   Carmel Valley Seminar for Industry Executives Sponsored by 3M Company

Peter E. Hart: "Intelligence: Natural and Artificial"

February 20, 1975   Special Systems Projects Office, U.S. Navy, Washington, D.C.

Peter E. Hart and Nils J. Nilsson: "Computer-Based Consultants"

February 20, 1975   UCLA Computer Science Department Seminar

Earl D. Sacerdoti: "The Nonlinear Nature of Plans"

March 21, 1975   Workshop on Cognitive Robotic Systems, California Institute of Technology, Pasadena, California

Richard E. Fikes: "An Overview of Robot Problem Solving"

Nils J. Nilsson: Panelist

Earl D. Sacerdoti: Panelist

APPENDIX

CHOOSING A TASK AREA FOR THE CBC PROJECT

A. Criteria

Given that our CBC system is to be an expert consultant about some complex electromechanical equipment of interest to the DoD, there is still the problem of selecting some specific class of equipment from among thousands of military items. We have acquainted ourselves with a few of the possible candidates, and we have looked into some of them quite thoroughly. Our prime consideration has been to select an item or class of items that meets certain criteria that we have adopted. We shall discuss these criteria first and then give some details about the equipment we have considered.

1. Relevance to DoD

The equipment should be of a type for which increased automation of maintenance will lead to substantial cost reduction and/or improved operation.

2. Amenability to the Use of Automation Methods

The equipment and its normal use should be such that computerized maintenance methods (if they can be developed) are feasible in an operational setting. It would be especially desirable if there were already beginning attempts to use computerized maintenance methods on the equipment.

3. Suitability for the Research Goals

The equipment must be such that the research performed using it leads to broadly useful results rather than results of interest for that equipment only.

4. Experimental Convenience

The equipment must be of a scale and complexity that permits us to experiment with it easily. It should not require inordinately elaborate test gear nor should it be too difficult or expensive to obtain.

5. Availability of Expertise

There should be a wealth of information about the equipment in the form of manuals, drawings, and nearby and accessible human experts.

B. Candidate Equipment

1. General

We have considered quite a few equipment items and have looked into four of them in some detail. Among the items we have only superficially thought about and rejected for various reasons before further study are: automatic manipulators (such as the Unimate arm), instruments such as gyroscopes, refrigeration systems, diesel engines for tanks and trucks, control-rod actuating mechanisms for nuclear reactors, large pneumatic compressor systems, torpedoes, and automatic transmissions.

The four items that survived initial screening are jeep engines, hydraulic flight control actuators, generator sets, and the steering mount for the AWG-10 radar for the A4 aircraft. We will comment on each of these in the next sections. We expect to select one of these items shortly.

2. Jeep Engines

Jeep engines are attractive because there is already an automatic system currently being field tested for diagnosing them. This system is the ATE/ICE being developed under contract to TACOM [46]. We have had initial conversations with personnel from TACOM, Frankford Army Arsenal, and RCA who have been involved in this project, and these have been on the whole quite encouraging. We would be able to build on the existing technology, adding more flexible communications and more powerful diagnostic procedures. There is a good chance that we would be able to obtain a copy of the ATE/ICE system.

A jeep would be reasonably acceptable from the experimental convenience point of view, except that running the engine inside wwould require special venting of the exhaust. It scores well on all of the other criteria.

3. Hydraulic Flight Control Actuators

On every commercial and military jet airplane there are several hydraulic actuators moving flight control surfaces such as elevators and ailerons. These are moderately complex devices that are nontrivial to troubleshoot.

We have not yet received official permission to visit the Naval Air Rework Facility at Alameda Naval Air Station to talk to the appropriate maintenance people, but in the meantime we visited the maintenance headquarters of United Airlines. There we learned of the variety of different actuators for commercial aircraft and something about the maintenance problems involved.

Our main concern about working with one of these actuators is experimental convenience. Some of these considerations include the skin irritating properties of the hydraulic fluid used in commercial actuators (military actuators are said to be better in this regard), the problem of fluid atomizing, and the need for several kinds of special sensors--e.g., flow meters, manometers, force transducers, position transducers.

There is a possibility that we could use United Air Lines' existing facilities, and this would solve some of the experimental problems.

4. Generator Sets

DoD uses a wide variety of generator sets ranging from small portable field units to large power systems. All of the newer ones are either turbine or diesel powered. A complete medium-sized system such as turbine powered EMU-30 (30 kW) involves complex electronic, electrical, and mechanical subsystems.

We visited the generator liasion office at McClellan Air Force Base in California to learn more about the characteristics of generators and their maintenance problems. Our conclusions are that a system could be chosen of the right level of difficulty, but there are some severe experimental convenience problems: they are very noisy.

5. The AWG-10 Steerable Radar Mount

We discussed the steerable radar mount for the AWG-10 radar used in the A4 aircraft at Miramar Naval Air Station. This is a complex device containing motors and servos to aim the radar antenna. It is of about the right size physically and would be optimal from an experimental convenience point of view.

We have some concern that it might not represent a maintenance problem of sufficiently general interest, and we have also experienced certain difficulties already in getting the appropriate Navy publications describing it.

REFERENCES

1. B. G. Buchanan, G. Sutherland, and E. Feigenbaum, "Heuristic DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry," Machine Intelligence, Vol. 4, B. Meltzer and D. Michie, eds, pp. 209-254 (American Elvesier Publishing Company, New York, New York, 1969).

2. B. G. Buchanan and J. Lederberg, "The Heuristic DENDRAL Program for Explaining Empirical Data," Proceedings IFIP Congress, Vol. 71, Ljubljana, Yugoslavia, 1971; also Stanford University AIM 141.

3. Joel Moses, "Symbolic Integration: The Stormy Decade," Proceedings ACM 2d Symposium on Symbolic and Algebraic Manipulation, S. R. Petrick, ed., Los Angeles, California (March 23-25, 1971).

4. A. L. Samuel, "Some Studies in Macnine Learning Using the Game of CHECKERS II--Recent Progress," IBM J. Res. Dev., Vol. II, 601-617 (1967).

5. R. Greenblatt et al., "The Greenblatt Chess Program," Proceedings AFIPS Fall Joint Comp. Conf., 801-810 (1967).

6. A. L. Zobrist and F. R. Carlson, Jr., "An Advice-Taking Chess Computer," Scientific American, Vol. 228, No. 6, pp. 92-105 (June 1973).

7. J. S. Brown, R. R. Benton, and A. G. Bell, "SOPHIE: A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting, (An Example of AI in CAI)," BBN Report No. 2790 (Bolt Beranek and Newman, Inc., Cambridge, Massachusetts, March 1974).

8. A. M. Collins, J. R. Carbonell, and E. H. Warnock, "Semantic Inferential Processing by Computer," Proceedings of the International Congress of Cybernetics and Systems, Oxford, England (August 1972).

9. T. Winograd, "Procedures as Representation for Data in a Computer Program for Understanding Natural Language," Technical Report AI TR-17, MIT, Cambridge, Massachusetts, 1971. Published as Understanding Natural Language (Academic Press, New York, New York, 1972).

10. D. E. Walker, "The SRI Speech Understanding System," IEEE Symposium on Speech Recognition, pp. 32-37 (April 1974).

11. Victor R. Lesser, et al., "Organization of the Hearsay II Speech Understanding System," IEEE Symposium on Speech Recognition, pp. 11-22 (April 1974).

12. W. A. Woods, "Motivation and Overview of BBN Speechlis: An Experimental Prototype for Speech Understanding Research," IEEE Symposium on Speech Recognition, pp. 1-10 (April 1974).

13. H. B. Ritea, "Voice-Controlled Data Management System," IEEE Symposium on Speech Recognition, pp. 28-31 (April 1974).

14. E. H. Shortliffe et al., "An Artificial Intelligence Program to Advise Physicians Regarding Antimicrobial Therapy," Computers and Biomedical Research, Vol. 6, 544-560 (1973).

15. N. J. Nilsson et al., "Plan for a Computer-Based Consultant System," draft report, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California (January 1974).

16. N. J. Nilsson et al., CBC Proposal to ARPA (in preparation) Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California (April 1975).

17. E. D. Sacerdoti, "A Structure for Plans and Behavior," forthcoming Ph.D. thesis, Stanford University, Stanford, California.

18. R. Reboh and E. D. Sacerdoti, "A Preliminary QLISP Manual," Technical Note 81, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California (August 1973).

19. R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence, Vol. 2, pp. 189-208 (1971).

20. D. G. Bobrow and B. Raphael, "New Programming Languages for Artificial Intelligence," Computing Surveys, Vol. 6, No. 3 (September 1974).

21. E. D. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," Artificial Intellience, Vol. 5, No. 2, pp. 115-135 (1974).

22. G. J. Sussman, "A Computational Model of Skill Acquisition," Technical Note AI TR-297, Artificial Intellience Laboratory, MIT, Cambridge, Massachusetts (August 1973).

23. E. D. Sacerdoti, "The Nonlinear Nature of Plans," Stanford Research Institute, Menlo Park, California, Artificial Intelligence Center Technical Note 101, January 1975. or Proceedings IJCAI 1975.

24. I. C. Braid, "Designing with Volumes" (Cantab Press, Cambridge, England, 1973).

25. "An Introduction to PADL," Production Automation Project, University of Rochester (December 1974).

26. M. E. Engeli, "A Language for Three-Dimensional Graphics Applications," International Joint Computing Symposium, Davos, Switzerland, American Elsevier, pp. 459-466 (September 1973).

27. G. J. Agin and T. O. Binford, "Computer Description of Curved Objects," Third International Joint Conference on Artificial Intelligence, Stanford, California (August 1973).

28. B. B. Baumgart, "Geomed--A Geometric Editor," Stanford Artificial Intelligence Laboratory, Stanford, California, Memo No. AIM-232 (May 1974).

29. R. O. Duda and P. E. Hart, Chapter 10, "Pattern Classification and Scene Analysis" (John Wiley & Sons, New York, 1973).

30. G. Agin and D. Nitzan, "An Algorithm for Finding the Outline of a Polyhedron Image," Perception Note Number 6, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California (May 1974).

31. D. Nitzan, "Procedure for Directing a Pointer to an Object Part," Perception Note Number 7, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California (June 1974).

32. J. Munson, "Chamfering," SRI Artificial Intelligence Center, Informal Communication.

33. D. V. McDermott and G. J. Sussman, "The Conniver Reference Manual," AI Memo No. 259, Project MAC, Massachusetts Institute of Technology, Cambridge, Massachusetts (May 1972).

34. C. Hewitt, "Procedural Embedding of Knowledge in Planner," Proceedings of IJCAI, London (September 1971).

35. J. F. Rulifson, J. A. Derksen, and R. J. Waldinger, "QA4: A Procedural Calculus for Intuitive Reasoning," Technical Note 73, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, Californi (November 1972).

36. D. E. Walker, "Speech Understanding Research," Annual Report, Project 3804, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California (April 1975).

37. N. J. Nilsson et al., "Artificial Intelligence--Research and Applications," Progress Report, prepared for ARPA, Contract DAHC04-72-C-0008, Stanford Research Institute, Menlo Park, California (May 1974).

38. W. H. Paxton, "Parsing Spoken and Written English," thesis in preparation, Stanford University, Stanford California.

39. R. F. Simmons, "Semantic Networks: Their Completion and Use for Understanding English Sentences," in "Computer Models of Thought and Language," Schank and Colby, eds. (W. H. Freeman & Co., San Francisco, California, 1973).

40. S. C. Shapiro, "A Net Structure for Semantic Information Storage, Deduction, and Retrieval," Advance Papers, Second International Joint Conference on Artificial Intelligence, pp. 512-523 (1971).

41. D. E. Rumelhart and D. A. Norman, "Active Semantic Networks as a Model of Human Memory," Advance Papers, Third International Joint Conference on Artificial Intelligence, Stanford, California, pp. 450-457 (1973).

42. G. Hendrix, "Partitioned Networks for the Mathematical Modeling of Natural Language Semantics," dissertation in preparation, Department of Computer Sciences, University of Texas, Austin, Texas.

43. W. Woods, "Transition Network Grammars for Natural Language Analysis," Communications of the Association of Computing Machines, Vol. 13, No. 10, pp. 591-606 (1970).

44. S. E. Fahlman, "A Planning System for Robot Construction Tasks," in Artificial Intelligence, Vol. 5, No. 1, pp. 1-49 (Spring 1974).

45. H. Y. Chang, E. Manning, and G. Metze, "Fault Diagnosis of Digital Systems (Wiley-Interscience, New York, 1970).

46. N. A. Teixeira and P. Bokros, "Current Status and Activities in Automative ATE," presented at IEEE International Convention and Exposition (March 26-29, 1974).

47. H. R. Warner, A. F. Toronto, and L. G. Veasy, "Experience with Bayes' Theorem for Computer Diagnosis of Congenital Heart Disease," Annals New York Academy of Science, vol. 115, pp. 558-567 (1964).

48. G. A. Gorry and G. O. Barnett, "Experience with a Model of Sequential Diagnosis," Computers and Biomedical Research, Vol. 1, pp. 490-507 (1968).

49. J. A. Jacquez, ed., "Computer Diagnosis and Diagnostic Methods," Proceedings of the Second Conference on the Diagnostic Process, University of Michigan (Charles C. Thomas, Springfield, Illinois, 1972).

50. E. H. Shortliffe, "MYCIN: A Rule-Based Computer Program for Advising Physicians Regarding Antimicrobial Therapy Selection," Artificial Intelligence Laboratory, Memo AIM-251, Stanford University, Stanford, California (October 1974).

51. E. H. Shortliffe and B. G. Buchanan, "A Model of Inexact Reasoning in Medicine," Stanford University, Stanford, California (July 1974).

52. J. Zlotnick, "Bayes' Theorem for Intelligence Analysis," in Computer Diagnosis and Diagnostic Methods, J. A. Jacquez, ed., pp. 180-190, Proceedings of the Second Conference on the Diagnostic Process, University of Michigan (Charles C. Thomas, Springfield, Illinois, 1972).

53. D. H. Gustafson et al., "Wisconsin Computer Aided Medical Diagnosis Project--Progress Report," in Computer Diagnosis and Diagnostic Methods, J. A. Jacquez, ed., pp. 255-278, Proceedings of the Second Conference on the Diagnostic Process, University of Michigan (Charles C. Thomas, Springfield, Illinois, 1972).

54. J. M. Tenenbaum, "On Locating Objects by Distinguishing Features in Multisensory Images," Computer Graphics and Image Processing, Vol. 2, No. 3, p. 308 (December 1973).

55. Second International Joint Conference on Pattern Recognition, Copenhagen, Denmark (August 13-15, 1974).

56. J. M. Tenenbaum et al., "Research in Interactive Scene Analysis," prepared for NASA, Contract NASW-2086, Stanford Research Institute, Menlo Park, California (March 1975).

57. P. E. Hart et al., "Artificial Intelligence--Research and Applications," prepared for ARPA, Contract DAHC04-72-C-0008, Stanford Research Institute, Menlo Park, California (December 1972).

58. J. M. Tenenbaum, "MSYS: A System for Reasoning About Scenes," Annual Report to ONR, Contract No. N0014-71-C-0294, Stanford Research Institute, Menlo Park, California (in preparation).

59. R. O. Duda, "Some Current Techniques for Scene Analysis," Technical Note 46, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California (October 1970).

60. A. Guzman, "Analysis of Curved Line Drawings Using Context and Global Information," Machine Intelligence, Vol. 6, B. Meltzer and D. Michie, eds., pp. 325-376, Edinburgh University Press, Edinburgh, Scotland (1971).

61. D. G. Waltz, "Generating Semantic Descriptions from Drawings of Scenes with Shadows," A.I. TR-271, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts (August 1972).

62. F. P. Preparata and S. R. Ray, "An Approach to Artificial Non-Symbolic Cognition," Information Science, Vol. 4, pp. 65-66 (1972).

63. Y. Y. Yakimovsky, "On the Recognition of Complex Structures: Computer Software Using Artificial Intelligence Applied to Pattern Recognition," Proceedings of the Second International Joint Conference on Pattern Recognition, pp. 345-353, Copenhagen, Denmark (August 1974).

64. A. Fischler and R. A. Elschlager, "The Representation and Matching of Pictorial Structures," IEEE Trans. Computers, Vol. C-22, pp. 67-72 (January 1973).

65. N. J. Nilsson, "Problem Solving Methods in Artificial Intelligence," (McGraw-Hill Book Co., New York, 1971).

66. J. M. Tenenbaum et al., "An Interactive Facility for Scene Analysis Research," Technical Note 87, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California (January 1974).

67. B. M. Wilber and E. D. Sacerdoti, "A QLISP Reference Manual," Artificial Intelligence Center Technical Note, Stanford Research Institute, Menlo Park, California (in preparation).

68. W. Teitelman, "INTERLISP Reference Manual," Xerox Palo Alto Research Center, Palo Alto, California (October 1974).

69. J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," J. ACM, Vol. 12, No. 1 (January 1965).

70. D. Luckham, "The Resolution Principle in Theorem-Proving," in N. Collins and D. Michie (eds.) "Machine Intelligence 1" (American Elsevier Publishing Co., Inc., 1967).

71. R. de la Briandais, Proceedins Western Joint Computer Conference, Vol. 15 (1959).

72. E. Fredken, "Trie Memory," C. ACM, Vol. 3 (1960).

73. D. E. Knuth, "The Art of Computer Programming," Vol. 3, "Sorting and Searching" (Addison-Wesley, 1972).